NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DATA BASE MANAGEMENT SYSTEM
FOR
MICROCOMPUTERS

by

Amrun Senan
and
Timbul Maruap Sihombing

December 1979

Thesis Advisor: F. Burkhead

Approved for public release; distribution unlimited.

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A085 036
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle) Data Base Management System for Microcomputers	5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; December 1979
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Amrun Sehan and Timbul Maruap Sihombing	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California	12. REPORT DATE December 1979
	13. NUMBER OF PAGES 111
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report) Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

microcomputer, UCSD Pascal, relational data base, query language, data separation, data independence

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

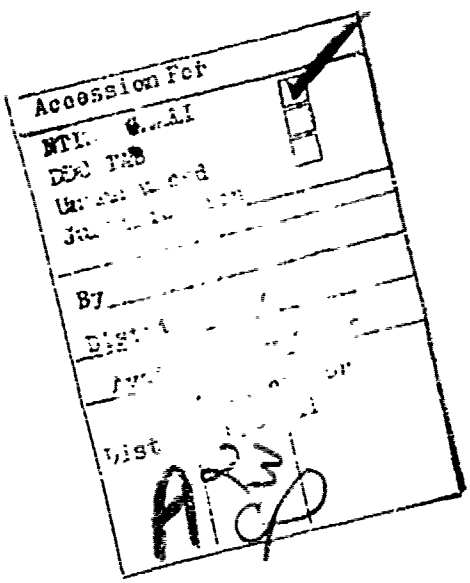
Many of the existing data base management systems have been developed for large applications such as big business. However other applications such as small businesses can also benefit from the managerial information which can be provided by a computer data base. This thesis develops a stand-alone data base management system using a microcomputer with floppy disk auxiliary storage and the UCSD Pascal software package. This system has the capability to create, update, delete and insert information, and

DD FORM 1 JAN 73 1473 (Page 1) EDITION OF 1 NOV 65 IS OBSOLETE S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

251450

to respond to user inquiries. Because of the limited storage capacity and relatively slow access speeds of floppy disks, the system will only satisfy small applications. However, the advent of compatible hard disk systems for microcomputers will enable the system to be used for significantly large applications.



Approved for public release; distribution unlimited

DATA BASE MANAGEMENT SYSTEMS FOR MICROCOMPUTERS

by

Amrun Sehan
Major, Indonesian Navy
B.S., Indonesian Naval Academy, 1967
M.S., Indonesian Naval Institute of Science, 1976

Timbul Maruap Sihombing
Major, Indonesian Army
B.S.,Padjadjaran University, Bandung, Indonesia, 1965

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1979

Authors: _____

_____

Approved by: _____
Thesis Advisor

_____
Second Reader

_____
Chairman, Department of Computer Science

_____
Dean of Information and Policy Sciences

ABSTRACT

Many of the existing data base management systems have been developed for large applications such as big business. However other applications such as small businesses can also benefit from the managerial information which can be provided by a computer data base. This thesis develops a stand-alone data base management system using a microcomputer with floppy disk auxiliary storage and the UCSD Pascal software package. This system has the capability to create, update, delete and insert information, and to respond to user inquiries. Because of the limited storage capacity and relatively slow access speeds of floppy disks, the system will only satisfy small applications. However, the advent of compatible hard disk systems for microcomputers will enable the system to be used for significantly larger applications.

TABLE OF CONTENTS

I. INTRODUCTION

It might be said that this is the data base era in computer technology. Data base processing has grown in significance among computer scientists and also among managers of organizations. The capacities of on-line data files have grown rapidly. As capacities go up, the cost per bit of storage comes down. This situation motivates data base designers to continue their efforts to obtain better data base systems.

An important consideration in data base design is to store data in such a way that it can be used for a wide variety of applications. By doing so, the data can be changed quickly and easily. To achieve the flexibility of data usage that is essential in most commercial situations, two aspects of data base design are important. First, the data should be independent of the programs which use it, so that it can be modified without the programs being changed (data independence). Second, it should be possible to interrogate and search the data base without the lengthy operation of writing programs in conventional programming languages. For this purpose, data base query languages are used. If the structure of the data base is independent of the program that

accesses it, then it is possible to develop an English-like, non-procedural, query language. The relational model supports data independence better than other models, and therefore, is the most appropriate among the existing data base models for development of a query language.

It is very difficult to design a data base which performs in an optimal fashion. There are many different ways in which data can be structured and each has its own advantages and disadvantages. Different users have different requirements. It is hardly possible to satisfy all of the users with one type of data organization. That is why trade-offs are frequently made. Examples include trade-offs between storage and time utilization, and between response time and complexity of data structure.

The data base system developed in this thesis strives for low initial and operating costs. It uses a microcomputer as the main computer and floppy disks as secondary storage.

The widely-available UCSD Pascal software is compatible with most microcomputers using the CP/M operating system. The software includes such features as a screen-oriented text editor, an easy to use file system, and a library manager. The system supports interactive programs and allows a program to be partitioned into separately compilable units. For these

reasons the UCSD Pascal software package was chosen as the underlying software support system [Ref.1].

The data is stored in the data base in the form of records. hence, a special program was developed to be used to create a new data base.

The data base system storage structure uses the separation technique. This technique consists of storing the data separately from the relationships. The advantages of this approach are faster data retrieval, more complete data independence and economical use of storage [Ref.2]. More details can be seen in chapter 6.

The data base management system developed in this thesis is a miniature version of data base management systems typically available on larger systems.

II. BACKGROUND

A. THE NECESSITY OF A DATA BASE MANAGEMENT SYSTEMS

Computer-based systems have become important tools for retrieving timely and accurate information. A data base system is expected to provide its user with the required information within a specified time. Most of today's data base management systems, were developed to manage large data bases. For small systems, these data base management systems are too expensive to implement.

Organizations within the Indonesian Armed Forces represent examples of organizations which need smaller data base management systems with reasonable operating and maintenance costs. There are many lower level organizations which need to provide fast information to the higher headquarters. They need to be equipped with local computerized systems which can provide fast and accurate information. By maintaining data in a data base system, accurate and up-to-date information can be maintained.

B. MICROCOMPUTER OVERVIEW

1. Introduction

In general, the term "microcomputer" can be defined as a stored program computer comprising memory and input/output circuits together with a microprocessor CPU [Ref.3]. Microcomputers have attracted many people because of several advantages over larger computers. First, microcomputers can be used for a wide range of specific applications. Second, microcomputers are powerful, reliable, and inexpensive. They can operate effectively in environments where older computers would fail. Most off-the-shelf microcomputers operate at room temperature and require no special air conditioning or power supplies.

2. Perspective On Microcomputers

The major disadvantages of existing data base systems are initial cost and high operating costs. The use of a microcomputer system should be considered as an alternative, obtaining a data base system of smaller size, but with lower operating costs. Recent developments in microcomputer technology have already provided them with the hardware capabilities to retrieve and update information. For microcomputers which have no such hardware the capabilities

can be implemented in software. Without these capabilities the development of a data base system is hardly possible. Floppy disks are currently the principle auxiliary storage devices used with microcomputer systems. Their physical organization is similar to hard disks, however, their storage capacities and access speeds are significantly less.

The use of a microcomputer as a stand-alone data base management system with an English-oriented query language, and UCSD Pascal as supporting software, seems feasible.

C. UCSD PASCAL

UCSD Pascal was chosen as the underlying software support system for the following reasons.

1. The software includes such features as a screen-oriented text editor, a useful file system, and a library manager. The system supports interactive programs.

2. The UCSD Pascal compiler is a one-pass recursive descent compiler. It generates code files to run directly on the Pascal interpretive machine. When compile-time errors occur, the user may optionally

return directly to the screen-oriented editor with the cursor at the position in the file where the error was detected.

3. Segment Procedure allows the user to partition a large program into several segments. Each segment is compiled separately. The Linker program is then used to link the separate segments together to produce one large code file. This procedure enables large programs to be run in the relatively small main memory.

4. The SETUP program enables the users to reconfigure the UCSD Pascal operating system to suit his or her equipment or taste. This is necessary when the system is used with different terminals or different machine configurations. SETUP enables the user to make these changes quickly and easily.

III. OVERVIEW OF A DATA BASE SYSTEM

A. INTRODUCTION

The term "data base" is still not accepted with a standard meaning. However it is to some extent accepted as conveying a more sophisticated concept than the older term "file". Confusion has arise over the meaning of the data base. Users tend to look upon a database as the aggregate of data from which they can make decisions.

The following definition of a data base is due to [Ref.2].

> "A data base is a collection of interrelated data stored together with the minimum redundancy to serve one or more applications in an optimal fashion. The data is stored so that they are independent of programs which must use the data. A common and controlled approach is used in adding new data and in modifying and retrieving existing data within the data base."

Data processors have tried to develop data bases without realizing the magnitude of the task. In reality, most of the today's data bases serve a limited set of applications.

In designing a data base system there are many facts that should be considered. The following are among the primary objectives of data base organizations.

1. It should make applications development easier, cheaper, faster, and more flexible.
2. The data should have multiple uses.
3. Data independence.
4. Clarity. Ease of understanding what data is available to the users.
5. Flexible usage. Data can be used in flexible ways with different access paths.
6. Spontaneous requests for data can be handled easily by means of a high level query language or report generation language.
7. Change is easy.
8. Low cost.
9. Accuracy and consistency.
10. Privacy

Of course we cannot expect all of these objectives will be gained in the optimum fashion at the same time. Depending on the user's requirements, the data base is designed to achieve their primary objectives. The primary objective of

this thesis is to design a data base system for small applications with low initial and operating costs.

B. DATA BASE SYSTEM COMPONENTS

A data base system involves three components:

1. User or application programs
2. The data base system (programs for accessing the data base.)
3. The data base itself

User programs are programs through which the users interact directly with the system. These programs are written in a query language. Applications programs interact with the system via a language such as COBOL or PL/I. The data base system is a set of computer programs, that operate on the data base in accordance with the user's commands.

C. DATA BASE STRUCTURE

For the present discussion, data structuring may be defined as the computerized representation of the relationship between distinct data items or data groups. The

following is an example. For a given school there are two basic sets of information. These sets are student data and course data. See figure [2-1]. Data on each student is maintained as a record and the collection of these records becomes the student file. Likewise records on each course are collect into the course file. These files are considered as the primitive data bases. To obtain the names of students which have been scheduled into classes, the relationships between data groups of these two files are required. Figure [2-2]. The school data base becomes more complex as additional record types are added. For example, a teacher file is added. The data base now should have relationships between the teachers and students and between the teachers and the courses.

Recalling from the definition above, the process of structuring a data base is a means of mapping the different data types. This mapping can be done using physical linkage or by inversion. Physical linkage implies that addresses of related records are stored within records themselves so that linkage "paths" exist within and among physical files. Inversion is the use of an 'inverted file' which contains the locations of the data base records [Ref.4]. An inverted file can be defined as a file which contains the entity identifiers associated with the values of certain attributes.

D. RELATIONAL DATA BASE APPROACH

In today's world, users (i.e managers) are more concerned with the information content of their data rather than its representation. Managers tend not to be bothered by bits, pointers, arrays, lists, etc., which may be used to represent information. Rather, they desire 'independence' from implementation details. In the relational model, information is represented at the user interface by data values only. User requests become free of any dependence on internal representation and hence may be framed in a high-level non-procedural language [Ref.5]. At the same time the system becomes free to choose any physical structure for storage of data and to optimize the execution of given requests. These characteristics are important for the use of a microcomputer as a main computer due to limited size of storage and slowness of access inherent with floppy disks. Data independency also enables the development of English-like non-procedural query languages.

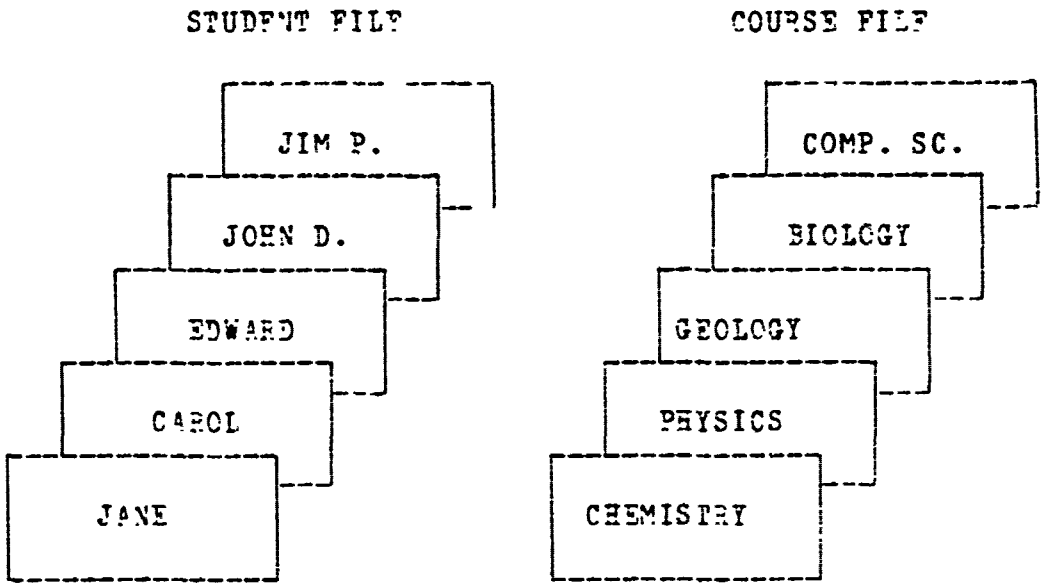


Fig. 2-1. School data base.

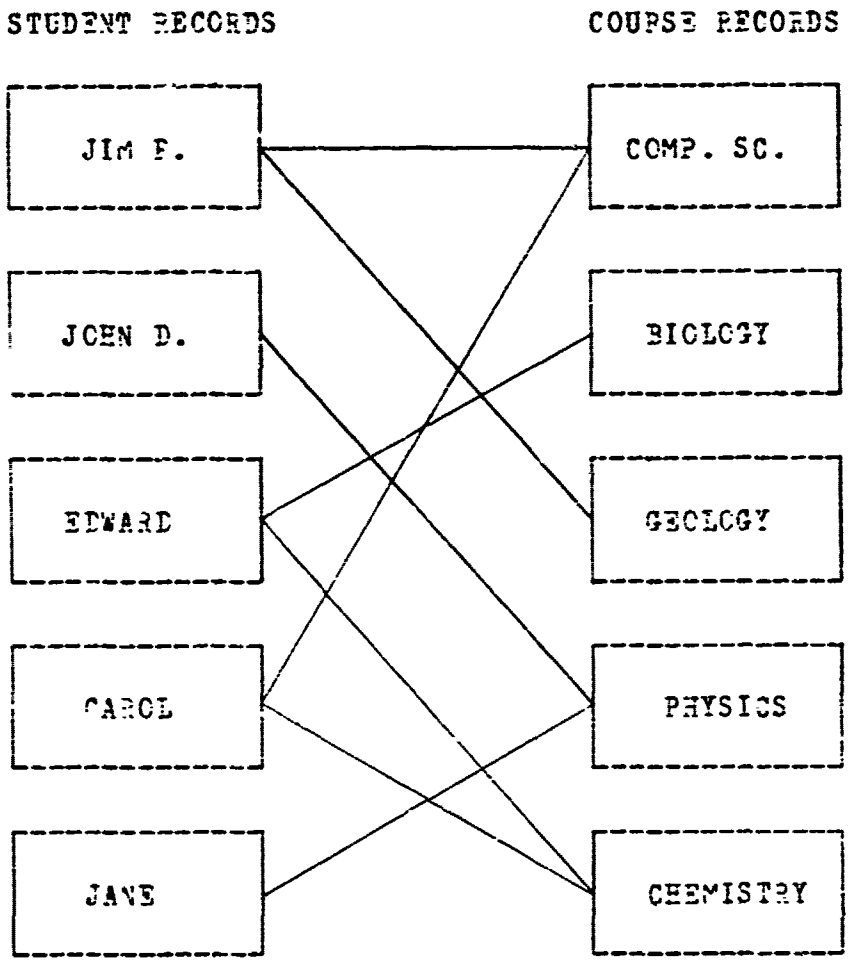


Fig. 2-2. STUDENT-COURSE Relationships.

IV. SYSTEM REQUIREMENTS

A. HARDWARE REQUIREMENTS

The system developed in this thesis uses an ALTOS microcomputer (Z-80 plus 64K memory) with two single-density floppy disk drives and the UCSD Pascal software system. Each floppy disk has a capacity of 256K bytes. The 64K memory size is considered to be adequate for storing the operating system, programs being executed and for working space. Certainly, the larger the size of the memory, the more suitable it becomes for the data base system implementation. The more significant limitation is in the floppy disk capacity. 512K bytes of storage is too small to hold the data base and the data base system programs. However, additional floppy disks, higher densities, and eventually, hard disks will greatly enhance the storage capacity.

The UCSD Pascal system is compatible with most microcomputers using the CP/M operating system. CP/M is the operating system used for Intel 8080 and Z-80 microprocessors. CP/M provides a general environment for program construction, storage, and editing, along with assembly and program check-out facilities. In general, it needs at least 16K memory with up to four disk drives.

However, Digital Research has recently developed CP/M version 2.0, wich can provide data management for up to 128 megabytes [Ref.6].

B. SOFTWARE SUPPORT

1. Introduction

To support the implementation of a data base system, the system needs an operating system, a data base creator, and the capabilities to create, delete and insert, update, and retrieve data. The query language is used by the user to interact with the data base. A help command is necessary to assist the users in obtaining information about the logical structure of the data base and to provide on-line information on how to use the data base system.

2. Operating System

The operating system was provided by UCSD Pascal. It is required to support the file structure that will allow dynamic allocation of file space, yet permit both sequential and random file access. The UCSD Pascal operating system can support the normal internal operations necessary for the microcomputer system to interface with standard peripheral

equipment (e.g. printer, CRT, disk, etc.).

3. Data Base Management System

A data base management system is a set of programs that operate on the data base in accordance with the user's commands. It has two major functions, i.e., data organization and data access.

After the user issues a request using the query language, the data base management system intercepts the request, and interprets it. Finally, it performs the operations on the data base. The data base management system developed in this thesis has three major subprograms: a parser, an interpreter, and a data base creator. The parser analyzes the syntax of the commands received from the terminal using the top-down parsing method. If the command is syntactically correct, it is coded into a table. The interpreter then takes the codes from the table, interprets them, and then executes the command. If the syntax is not correct, an error message is displayed at the terminal. The user can then repeat the request with the correct command.

The data base creator is used to create a new data base, or add data to an existing data base from the terminal. The text editor developed by UCSD Pascal cannot be used to

create data files for the data base. This is because the required files are organized as sequences of encoded record structures. The UCSD Pascal text editor can only read and write text files (sequences of characters). To create the data base, a program was developed which reads a record from the terminal and writes it to a file of the desired type.

4. Query Language

A query language is an English-like, self-contained data language. A self-contained data language is a complete programming language for both obtaining and manipulating data from a data base. The query language used in the developed system, is a SEQUEL-like query language [Ref.7,8,9]. It was developed to be able to retrieve data, delete data, modify existing data, and add new data.

Retrieve operations are represented syntactically as SELECT-FROM-WHERE blocks. For example, a command to get suppliers numbers and status of the suppliers in Paris would look like

```
SELECT S#,STATUS
FROM S
WHERE CITY = 'PARIS'.
```

The command is executed by finding and combining into a set all the rows where CITY = 'PARIS'. Then, using the vertical mapping for columns S# and STATUS on this set , the requested information can be obtained.

Deleting data, modifying existing data, and adding new data to the data base are the storage operations. The DELETE command is used to remove records from the data base, the UPDATE command is used for modifiyng existing data, and the INSERT command is used for adding new records to the data base.

5. Help command.

The help command is used to obtain information about the existing data base, i.e., its logical structure, the data type and status of each attribute (retrieval or non-retrieval) etc. It also provides on-line information on how to use the data base system.

C. SYSTEM OPERATIONS

After initialization, the data base system prompts the user with the following line.

Command: C'reate, H'elp, X(ecute

Using these commands, the user is able to create a new data base, to obtain information about the existing data base, or to retrieve and update the existing data base. The details concerning each command are provided in the user's manual (appendix C).

V. SYSTEM STORAGE STRUCTURE

A. INTRODUCTION

A data base management system is an effective managerial tool only if its response time is significantly better than existing manual systems.

To design such a data base management system using a microcomputer as the main computer, the arrangement of the on-line auxiliary storage structure must also be considered. Currently, floppy disks are the primary auxiliary storage devices found with microcomputers. Hopefully, the limitations on storage capacity and access time inherent in floppy disks will be overcome in the future by hard disk systems. The auxiliary storage structure developed in this thesis separates data and relationships. The technique is to develop and store the relationships separately from the data [Ref.2].

B. SEPARATING DATA AND RELATIONSHIPS

Loading the data base into main memory would provide better access time. However, this technique is unusual and often impossible to implement. This is especially true for

large data bases. It is also a problem for small data base systems in a microcomputer based systems, because of the limited amount of main memory.

The separation of data and their relationships enables the 'data base' to be loaded into the main memory in parts. Each data item value is stored once, and is given a serial number. The relationships are stored in terms of serial numbers. This saves storage space with those data-item types for which multiple data-items have the same value. Therefore, it is possible to load all the relationships, which represent the data base in terms of serial numbers, into main memory. All accesses involving these relationships can then be improved.

Another alternative is the possibility of loading only the needed relationships. It is not necessary to load all of the relationships in a data base if the user only deals with a particular relation.

Separation can be done at the segment level, i.e., groups of attributes, or it can be done at the data item level, i.e., individual attributes, where relationships between data values are stored separately from those values.

There are, now, two kinds of files; the data files and

the relationships files. Data files are the files that contain the data item values. Relationships files can be grouped into inverted and non-inverted files. The inverted relationships files are those that contain the entity-identifiers of the logical or user's files associated with the values of any retrieval attribute. Non-inverted relationships files are those which contain the relationships of all retrieval attributes.

The objectives of this approach are:

1. To make possible faster data retrieval.
2. To provide more complete data independence.
3. To save storage.

C. DATA FILES

1. Introduction

Data files are organized separately from relationships files. There are three methods which can be used to construct a data file. First, the data item values of each retrieval attribute can be stored in a separate file (fully separated). Second, the data item values of all the retrieval attributes of the same type are stored in one file (separated by segment). Third, the inverted relationships

files are embedded in the associated data files (partially separated). Each of these methods will now be considered in more detail.

2. Fully Separated File

The data item values of each retrieval attribute are stored in a separate file. See Fig. [5-1]. The data item values of non-retrieval attributes are included in the "entity-identifier file"1). The advantage of this method is the ease of data update. One data item can be inserted to, or deleted from, an attribute file2), without affecting any other file.

Each retrieval record can be accessed randomly, when the record number of the item within the attribute file is known. The record numbers of the retrieval data items are stored in the relationships files. Using the given data item as the key, the record number of this data item in the attribute file can be searched. This record number is used to find the value of the requested data item. By knowing its record number, the desired data item can be searched using the random access method.

---

1) The entity-identifier file is a file which contains entity-identifier attribute plus non-retrieval attributes.
2) The attribute file is a file which contains values for one attribute only.

3. Separated By Type

In this method, the contents of the same type attributes are stored in one file. However the record number of the first data item of each attribute which develops this file, must be maintained. These record numbers are stored at the beginning of the file, just before the first record, or they can be stored in a special file. See figure [5-2]. The record type can be of fixed length or variable length.

Before accessing the relationships file, the record number of the first data item of the attribute must be subtracted from the record number of the given data item. Then to the result of the subtraction, one must be added. The result of the addition is used as the record number of the given data item in the associated relationships file. The advantage of this method is in storage utilization. The disadvantage is that, if changes to the file are frequent, deletion and insertion of a large file will be time consuming and costly. This disadvantage can be reduced by storing the rarely changed attributes at the very top of the file and the frequently changed attributes at the bottom.

4. Partially Separated Files

The time used to move control from a data file to a

relationships file, and vice-versa, can be reduced if they are physically close together. This can be done by embedding the inverted relationships files into associated attribute files. The relationships files, now, are merely the inverted ones. See figure [5-3].

D. RELATIONSHIPS FILE

Relationships files are used to store the relationships within the logical file or user's file. The relationships files contain the record numbers of the related attributes only. There are two types of relationships files. One is the file which contains relationships of all retrieval attributes. This file is a non-inverted file. The other files are inverted files. Each inverted file contains the entity-identifiers of the logical or user's files, associated with the values of any retrieval attribute. Access to the relationships file is accomplished after obtaining the record number of the given data item. Using this record number, access to the associated inverted file will give the record number of all entity-identifiers which have relationships to the given data item. After obtaining the record number of these entity-identifiers, the relative record numbers of the desired data items can be accessed in the non-inverted relationships file. Finally, these record numbers are used to

access associated data files to obtain the requested information.

RELATIONSHIPS

SSN	NAME	DEPT#	SKILL	SAL
1	3	1	1	3
2	1	1	4	4
3	6	2	3	1
4	5	3	3	1
5	2	3	2	2
6	4	1	2	2

NON-INVERTED REL. FILE.

NAME	SSN
1	2
2	5
3	1
4	6
5	4
6	3

DEPT#	SSN
1	1
	2
	6
2	3
3	4
	5

SKILL	SSN
1	1
2	5
	6
3	3
	4
4	2

SAL	SSN
1	3
	4
2	5
	6
3	1
4	2

INVERTED RELATIONSHIPS FILES

DATA

no.	SSN	NON-INDEXED DETAILS
1	07642	
2	07643	
3	07650	
4	07668	
5	07670	
6	07671	

no.	NAME
1	ALBEY
2	ANDREWS
3	BLANAGAN
4	DALL
5	EDWARDS
6	FEINBURG

no.	DEPT#
1	119
2	210
3	220

no.	SKILL
1	Administr.
2	Fitter
3	Plumber
4	Secretary

no.	SAL
1	1000
2	1050
3	1200
4	1750

Fig. 5-1. Fully Separated Structure.

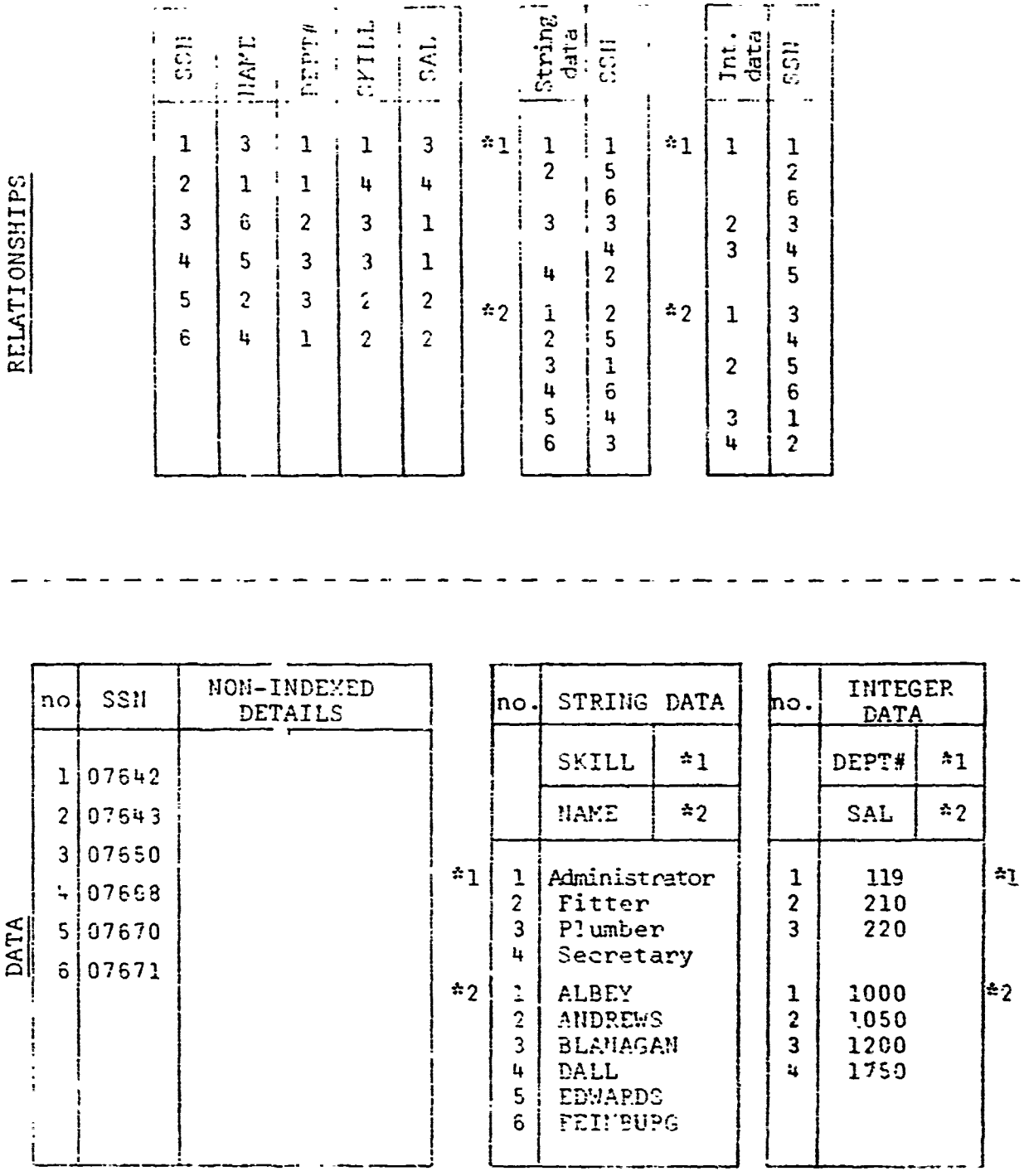


Fig. 5-2. Separated by Type.

SSN	NAME	DEPT#	SKILL	SAL
1	3	1	1	3
2	1	1	4	4
3	6	2	3	1
4	5	3	3	1
5	2	3	2	2
6	4	1	2	2

no.	SSN	NON-INDEXED DETAILS
1	07642	
2	07643	
3	07650	
4	07663	
5	07670	
6	07671	

no.	NAME	IDX
1	ALBEY	2
2	ANDREWS	5
3	BLANAGAN	1
4	DALL	6
5	EDWARDS	4
6	FEINBURG	3

no.	DEPT#	IDX
1	119	1
		2
		6
2	210	3
3	220	4
		5

no.	SKILL	IDX
1	Administrator	1
2	Fitter	5
		6
3	Plumber	3
		4
4	Secretary	2

no.	SAL	IDX
1	1000	3
		4
2	1050	5
		6
3	1200	1
4	1750	

Fig. 5-3. Partially Separated Structure.

VI. DATA BASE DESIGN

A. STRUCTURE ORGANIZATION

The data base structure developed in this thesis is a fully separated structure. The initial data base is created using a special program.

UCSD Pascal has the capability to access variables of type string, either of fixed or variable length. It also supports random access files. Therefore, all data item values and relationships are stored as records. Since the first record of a file is the 0th record, the physical record number is one less than the serial number.

Since the relationships files contain the record numbers of the data files, the organization then, is similar to the basic inverted organization, except that it uses a relation file (non-inverted relationships file) rather than the original data base records.

The advantage of this system is good performance in response to various types of queries involving the inverted attribute values without searching the whole file. Queries

like "Are there any personnel in department No. 102 with salary equal to 3000" can be answered easily by ANDing the lists of pointers (record numbers) corresponding to the access keys DEPT# and SAL in the query. It also provides more complete data independence. By separating the data, different applications will be able to have different views of the same data. It would be possible to change the applications without any changes in the stored data.

Another advantage is storage space saving. For illustration, suppose there are only six SKILL values. The file contains many secretaries, but the value 'SECRETARY' is stored only once in the data file. All the values 'SECRETARY' in the data base are stored in terms of the serial number of 'SECRETARY' in the non-inverted relationships files. In the relationships files, a SKILL value could be referred to with 3 bits. Using UCSD Pascal this value is stored as an integer in a 16-bit word.

The disadvantage is that it is more costly for the updating process because of the necessity of maintaining both the data and the relationships files. However, it is still better than the doubly chained tree or multilist organizations [Ref.12].

B. TFT DATA BASE ACCESS

The data base management system will handle all access to the data base. It intercepts the user's request, interprets it, and performs the necessary operations on the data base. It makes use of an access method to handle the details of physical access to the data base. This physical (lower level) access method is performed by the UCSD Pascal operating system.

There are four major programs which were developed to access the data base. CRTTT is a program which creates the initial data base. The PARSTR is a program which verifies the user's request and provides all information needed by the INTERPRETTR. The INTTRPRTTR is a program which performs the necessary data base manipulations. TTLP is a program which provides the user with information concerning the logical structure of the data base.

All the programs are written in UCSD Pascal. They were compiled separately and stored in the system library.

1. Create

The initial data base is created by invoking the CRTTE program. The data can be stored in the desired file in

record format, either of type integer, string, set of integers, or a combination of those. For convenience, all data files are stored in floppy disk drive No.1. The system disk (drive #) is used for the operating system, utilities, the data base programs, and user application programs.

2. Parser

This program first initializes the system for data base manipulation operations. Then it asks the user to describe the data base he or she needs access to. It accepts the user's request written in a SEQUEL-like query language. Error messages will be displayed at the terminal for any syntax error, and the user will be prompted to reenter his or her request. When correct input is received, the program generates a sequence table and a reference table. These tables will be used by the INTERPRETER.

3. Interpreter

In performing data base manipulations, the INTERPRETER follows the sequence provided in the sequence tables. The results of the action taken by the interpreter will be displayed at the user's terminal. Any file manipulation error, will be discovered by UCSD Pascal's system. The system will not recover from this type of error.

There are five major procedures in INTERPRETER.

a. CONDITION

This procedure gives the tuple number(s) which meet the condition (qualification) in the user's request.

b. QUERY

This procedure selects the desired attribute(s) in the selected tuple(s) as a result of (a) above.

c. UPDATE

Any changes on data and relationships files as the effect of the updating process, will be handled by this procedure.

d. INSERTION

This procedure performs any addition to the data base, i.e., addition to the data files, and addition or changes in the relationships files.

e. DELETE

This procedure deletes the data which the user wants to remove from the data files. This procedure also deletes or changes the values of the relationships files.

4. Help

This unit is intended to assist the user in finding information about the data base, i.e., the logical structure of the data base, and provides on-line information on how to use the data base system.

C. QUERY LANGUAGE

The SEQUEL-like query language as previously discussed is provided to the users to communicate with the data base. Syntax graphs and procedures are available in Appendix D.

VII. CONCLUSIONS

To solve the cost problems arising in the implementation of data base systems, particularly for small applications, the use of microcomputer systems should be considered as an alternative. The developed system has demonstrated that currently existing microcomputers, using UCSD Pascal as the system software and floppy disks as secondary storage, could be considered for small applications. Although the current auxiliary storage capacities are too small, additional floppy disks, higher densities and eventually, hard disks, hopefully can solve this storage problem.

The technique of storing data separately from their relationships enables the loading of all the relationships, at least the needed ones, into main memory. All accesses involving these relationships, can then be improved. It is also possible to obtain more complete data independence and to save storage.

APPENDIX A: USER'S MANUAL

A. INTRODUCTION

This data base management system is intended to provide the means for creating, retrieving, and updating a data base in a microcomputer based system supporting UCSD Pascal. It has three major functions.

1. Data base initialization.
2. Data base manipulation.
3. Providing helpful information on the data base.

This system is executed with the X(ecute command at the outermost command level of UCSD Pascal.

X(ecute <MICRODATA>

It displays three available commands associated with the three functions above. The prompt line

Command: C(reate, H(elp, X(ecute

will be shown on the screen. The user can select one of them. For example, by typing "X" the system will execute the data base manipulation function. With a SEQUEL-like query language

he or she can communicate with the data base.

Before working with this system, the user is expected to put the system disk in drive 0 and the data base disk in drive 1.

B. DATA BASE INITIALIZATION

For creating a data base initially a special program is used. By typing C(reate at the command level, a software unit named CREATE is invoked to serve that function. The system will automatically create the desired file in the disk that currently exists in drive 1. At this level, it is possible to delete a character on the current line without leaving the create mode by back-spacing over it. The following steps describe how to use the system for data base initialization:

1. By selecting C(reate, a message

"The system is ready to create a file.
Please enter your file name ------->"

will be displayed on the screen.

2. The user response would be

<file-name><CR>.

There are three categories of file names:

a. A data file name is identical to an attribute name.
   Example:

   NAME <CR>

b. An inverted relationships file name.
   Example:

   EMP.NAME

c. A non-inverted relationships file name is the name of the relation.
   Example:

   EMP <CR>

A string of up to 8 printed characters in which the first character is alphabet is acceptable. In case of an invalid file name, the user is asked to repeat.

3. A valid name will give the prompt:

   "Any non-retrieval attribute to be

entered in this file?Y/N.'

The user has a chance to enter his or her non-retrieval data-item value only in the file of the key attribute. Thus, the user may type 'Y' if he or she wants to or 'N' if he or she want to put them in another file.

4. A 'Y' will cause the next prompt:

"Please give its name."

The user response is like 2a. above.

5. The system asks for the data type.

'Data type? [INT'eger, STR'ing, SET.]"

INT'eger means the data are of type integer, STR'ing are of type string and SET are of type set of positive integers.

6. The user response would be

<data type><CR>

In the case of a non-retrieval attribute, only the type of its retrieval attribute is entered. The non-retrieval

attribute data value is considered to be of type string.

7. The next prompt will be displayed:

"Please enter your data.
=====>

If a non-retrieval attribute exists in the key attribute data file, the entering data will be interpreted alternately- i.e retrieval, non-retrieval, retrieval, and so on. The format for this is:

<retrieval-data><CR>
<non-retrieval-data><CR>

8. A control C will terminate the system with the following message.

"A file named ..... has been entered with .... records."

C. DATA BASE MANIPULATION

To communicate with the data base, a SEQUEL-like query language is used. The system intercepts user's request

written in that language, interprets it, and performs necessary operations for data manipulation purposes. Error messages will be displayed on any syntax error or data manipulation error, and the user is asked to repeat. I/O errors however, will be handled by UCSD Pascal. If I/O errors occur, the system must be reinitialized. At this level, the back space character is used to delete a character on the current line without leaving the execution mode.

The following are steps in using the system for data manipulation.

1. Type X'ecute at the command level. The following message will be displayed:

"The system is ready for data manipulation.
Please describe your data base."

The user user is expected to describe the data base i.e., all the relations and attributes where he or she wants to work on.

2. The next prompt:

' Relation ====\'

asks the user to describe his or her relation. The user response would be

<rel-name>(<att-name>{,<att-name>})<CR>.

Braces '{}' imply zero or more repetitions.

3. The system responds with the next prompt:

"Attribute ===>"

The user response would be

<att-name>,<status>,<type>{<CR><att-name>,<status>,<type>}
[',' or '.']<CR>

The ',' and '.' are optional. They are used for describing another relation and terminating the description respectively. Without ',' or '.', the prompt ===>" will be displayed for the next attribute.

4. Entering a ',' causes the prompt:

"Relation ===>"

and step 2 and 3 above have to be repeated.

5. The system's response on '.' is

'Preparation is completed.
Now you may give your command.
===>'

6. The system now is ready to intercept any user's request (command) written in a SEQUEL-like query language. Format:

<command>.<CR>

7. Results or outputs are displayed on the screen for each succesful data manipulation, followed by a "===>" which indicates that the system is ready for the next command.

8. Error messages will be displayed for syntax, data manipulation or I/O errors, either during data base description or manipulation. Then the user is asked to redescribe his or her data base or repeat his or her request. For example.

. FMP'EMP#,NAME,DEPT#,SKILL <CR>

will cause the message

Symbol ";" is needed.

. SELECT EMPNO FROM EMP .....

gives the message

"Undefined relation."

9. I/O errors are handled by UCSD Pascal. If they occur control is transferred to UCSD Pascal, the system quits and has to be reinitialized by

X(ecute <MICRODATABASE>

at the command level of UCSD Pascal. Then procedures from step 1 have to be followed.

10. Control-C terminates the data base manipulation function with the message:

" Data base manipulation has been completed. "

Control is transferred to UCSD Pascal system.

D. HELP FUNCTION

This system is also provided with the help function, which helps the user to obtain information about the currently existing data base. It will list all the existing relations and attributes including their status and types.

By selecting H(elp at the command level, the help function is performed and the information will be displayed. For example

Relation: EMP(EMP#,NAME,DEPT#,SKILL,SAL,ADDR)

EMP#,KEY,NBR

NAME,RETR,CH

DEPT#,RETR,NBR

SKILL,RETR,CH

SAL,RETR,NBR

ADDR,NONR,CH

Relation: CHILD(EMP#,CHNAME,SEX,AGE)

EMP#,CKEY,NBR

CHNAME,CKEY,CH

SEX,RETR,CH

AGE,RETR,NBR

Control-C terminates the help function and transfers

control back to the UCSD Pascal system.

APPENDIX B: QUERY LANGUAGE

A. INTRODUCTION

A stand-alone SEQUEL-like query language is used in this system. It is intended to provide the retrieval (SELECT) and storage (UPDATE, INSERT, DELETE) operations on the existing data base. Some of these have not been implemented completely. For creating a new data base or file, a special program is used. (See Data Base Initialization).

B. RETRIEVAL OPERATION

1. Simple Retrieval

examples:

a. Get all employees' numbers and names.

```
SELECT EMP#,NAME
FROM EMP.
```

b. Get all employees' skills.

```
SELECT UNIQUE SKILL
FROM EMP.
```

c. Get full detailed information of all employees.

```
SELECT *
FROM EMP.
```

2. Qualified Retrieval

Get the employees' numbers and names in department no. 100 with salary greater than 3000.

```
SELECT EMP#,NAME
FROM EMP
WHERE DEPT#=100 AND SAL > 3000.
```

3. Retrieval With Ordering.

Get the female children's name and age arranged in descending order of age.

```
SELECT CHNAME, AGE
FROM CHILD
WHERE SEX='FEMALE'
ORDER BY AGE DESC.
```

4. Retrieval Using Nested Mapping

Get all employees' names who have female children.

```
SELECT NAME
FROM EMP
WHERE EMP# IN
      SELECT EMP#
      FROM CHILD
      WHERE SEX='FEMALE'.
```

5. Retrieval Using Several Levels Of Nesting

Get the department name of the employees who have female children.

```
SELECT DEPTNAME
FROM DEPT
WHERE DEPT# IN
      SELECT DEPT#
      FROM EMP
      WHERE EMP# IN
            SELECT EMP#
```

```
FROM CHILD
WHERE SEX='FEMALE'.
```

6. Retrieval Using A Nested Mapping With Interblock Reference.

Get employees'' name who has never been in department number 100.

```
SELECT NAME
FROM EMP
WHERE 100 NOT IN
      SELECT DEPT#
      FROM DEPT
      WHERE EMP# = EMP.EMP#.
```

7. Retrieval Using A Nested Mapping With The Same Table Involved In Both Blocks.

Get the employee numbers of employees' who have children of the same age with at least one of the children of the employee with number 07672.

```
SELECT UNIQUE EMP#
```

```
FROM CHILD
WHERE AGE IN
      SELECT AGE
      FROM CHILD
      WHERE EMP# = '07672'.
```

8. Retrieval Involving Set Comparison

Get supplier names for suppliers who supply all parts.

```
SELECT SNAME
FROM S
WHERE (SELECT P#
       FROM SP
       WHERE S# = S.S#)
          =
      (SELECT P#
       FROM P).
```

9. Retrieval Involving MINUS

Get employees' numbers who have no children

```
(SELECT EMP#
 FROM EMP)

  MINUS

(SELECT EMP#
 FROM CHILD).
```

10. Retrieval Involving an Enumerated Tuple

Get employees' numbers who have the same skill and salary as employee number '07672'.

```
SELECT EMP#
FROM EMP
WHERE <SKILL,SAL> IN
      SELECT SKILL,SAL
      FROM EMP
      WHERE EMP# = '07672'.
```

C. STORAGE OPERATION

1.Update

a. Simple Update

Change the department number of employee numbered 07672 to 210

```
UPDATE EMP
SET DEPT#=210
WHERE EMP#='07672'
```

b. Repeated Update

Change the department number to 210 and the salary to 3000 of the employee numbered 07672

```
UPDATE EMP
SET DEPT#=210,
SET SAL =3000
WHERE EMP#='07672'.
```

c. Multiple Update

Suppose CHILD includes an extra attribute PREM (premium, say 10 percent of his/her parent's salary per child under 20 years old.) An increase on an employee's salary will increase his/her children's premium.

```
UPDATE EMP
SET SAL=300
WHERE EMP#='07672'.
```

```
UPDATE CHILD
SET PREM=3000
WHERE EMP#='07672' AND AGE <20.
```

2. Insertion

Add an employee named JONES, C.H. with employee's number 07800 in department 200 as the secretary with salary 3000.

```
INSERT INTO EMP
<'07800','JONES C.H',22,'SECRETARY',3000>.
```

3. Deletion

a. Qualified deletion

Delete CHILD if more than 20 year old.

```
DELETE CHILD
WHERE AGE > 20.
```

b. Unqualified deletion

Delete all children.

```
DELETE CHILD.
```

D. LIBRARY FUNCTIONS

1. Function In The SELECT Clause

a. Get the total number of employees.

```
SELECT COUNT(EMP#)
FROM EMP.
```

or

```
SELECT COUNT(*)
FROM EMP.
```

b. Get the total number of employees who currently have children.

```
SELECT COUNT(UNIQUE EMP#)
FROM CHILD.
```

2. Function In The Select Clause

With A Predicate

Get the total number of employees who have female children.

```
SELECT COUNT(EMP#)
FROM CHILD
WHERE SEX = 'FEMALE'.
```

3. Function In The Predicate

Get the employee's numbers who have the maximum salary.

```
SELECT EMP#
FROM EMP
WHERE SAL =
      SELECT MAX(SAL)
      FROM EMP.
```

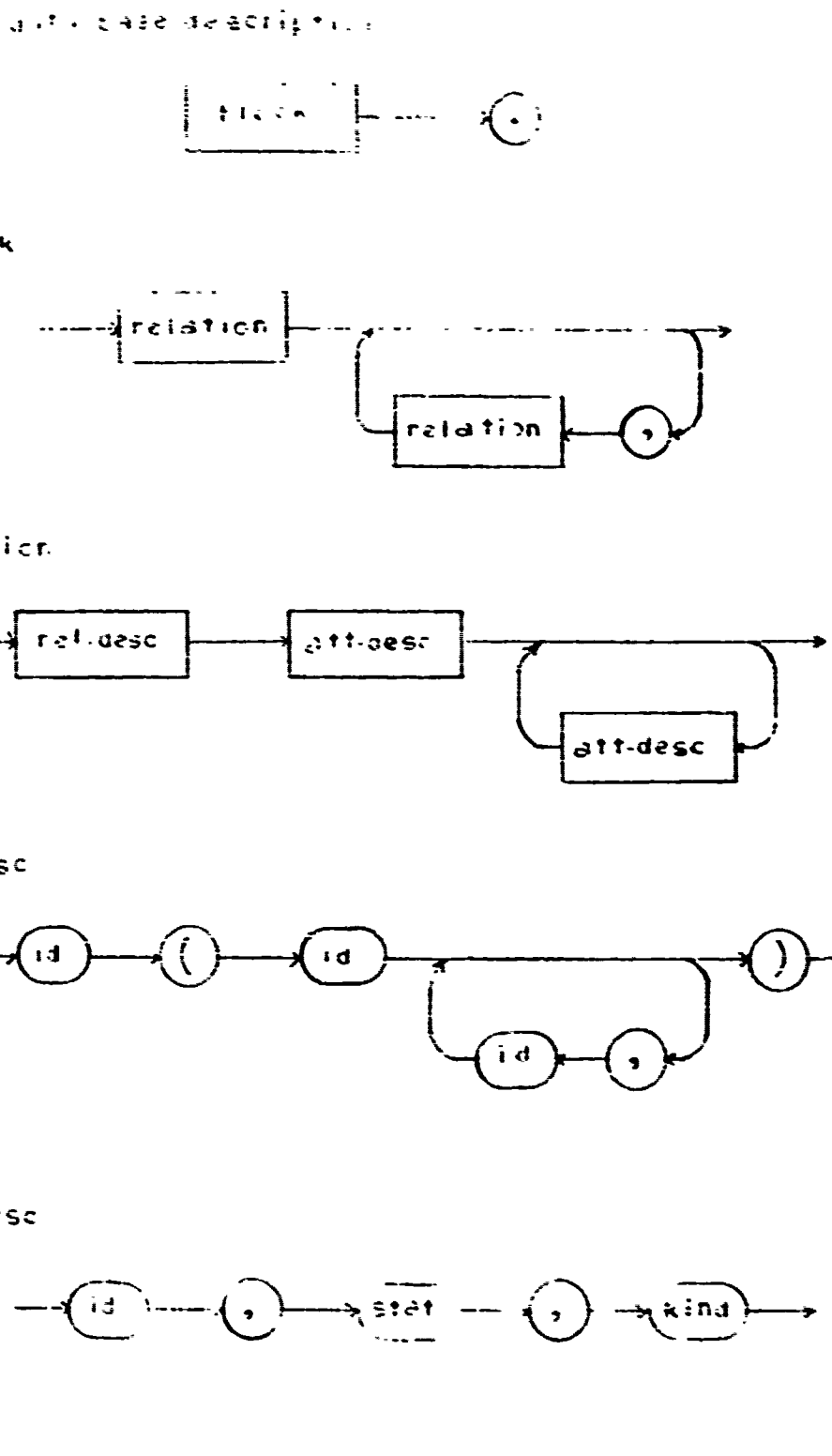


Fig. 4-1. Syntax graph for describing the data base.

APPENDIX C: GLOSSARY OF TERMS

1. Attribute. A term referring to a column of a relational file in a relational data base system.

2. Attribute file. A file which contains values for one attribute only.

3. Control-C. A control key used for terminating an operation by pushing CTRL and C buttons simultaneously.

4. CP/M. An operating system used for Intel 8080 and Z-80 microprocessors.

5. <CR>. Carriage return key.

6. Data independence. Immunity of applications to change in storage structure and access strategy, which implies that the applications concerned do not depend on any one particular storage structure and access strategy.

7. Entity. Item about which we store information. For example, employee's name, address, etc., are the entities.

8. Entity-identifier. A key which uniquely identifies an entity or data concerning that entity.

9. Entity-identifiers file. A file which contains the entity-identifier attribute plus non-retrieval attributes.

10. Inverted file. A file which contains the entity-identifiers associated with the values of certain attributes.

11. Inverted relationships file. A file which contains the values of a retrieval attribute and the entity-identifiers of the logical or user's file associated with the values of that retrieval attribute.

12. Non-inverted relationships file. A file which contains the relationships of all the retrieval attributes.

13. Prompt line. A display at the terminal which shows the current mode and the options available for that mode. Available in the UCSD Pascal system.

14. Query language. An English-like, complete programming language for both obtaining and manipulating data from a data base.

15. Relationships file. A file which contains the relationships of the related attribute files, in terms of serial numbers or record numbers.

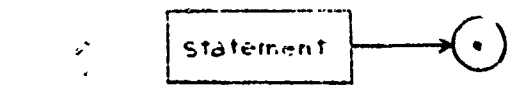
16. Separation technique. A technique to develop and store the relationships within the logical or user's file separately from the data.

17. UCSD Pascal. A software system, highly machine independent, used for stand-alone microcomputers or minicomputers.

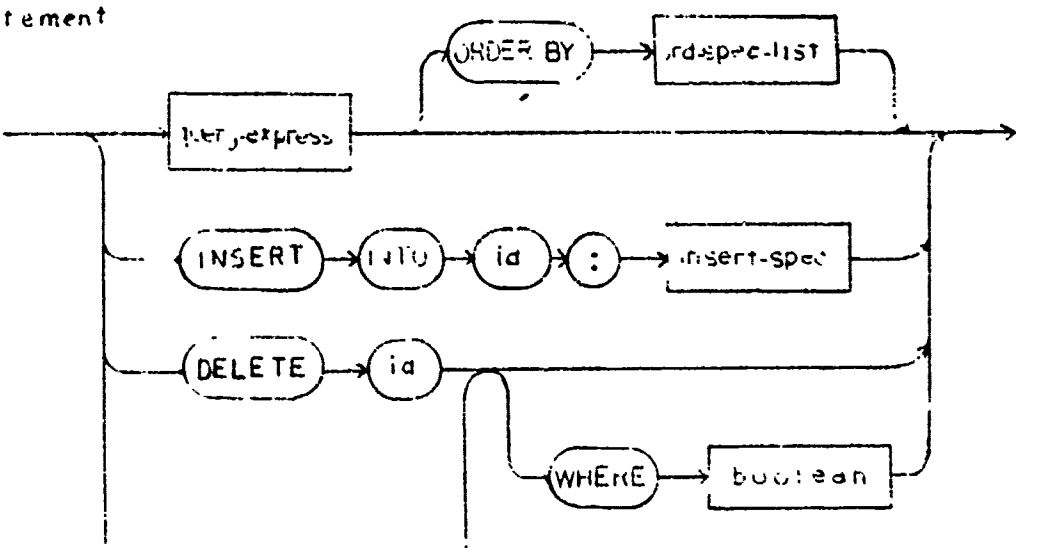
APPENDIX - I

SEQUEL-LIKE QUERY LANGUAGE SYNTAXGRAPH

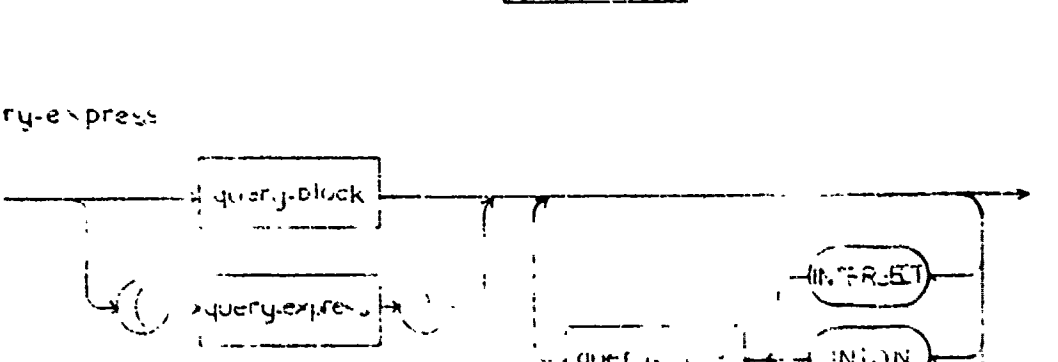
command

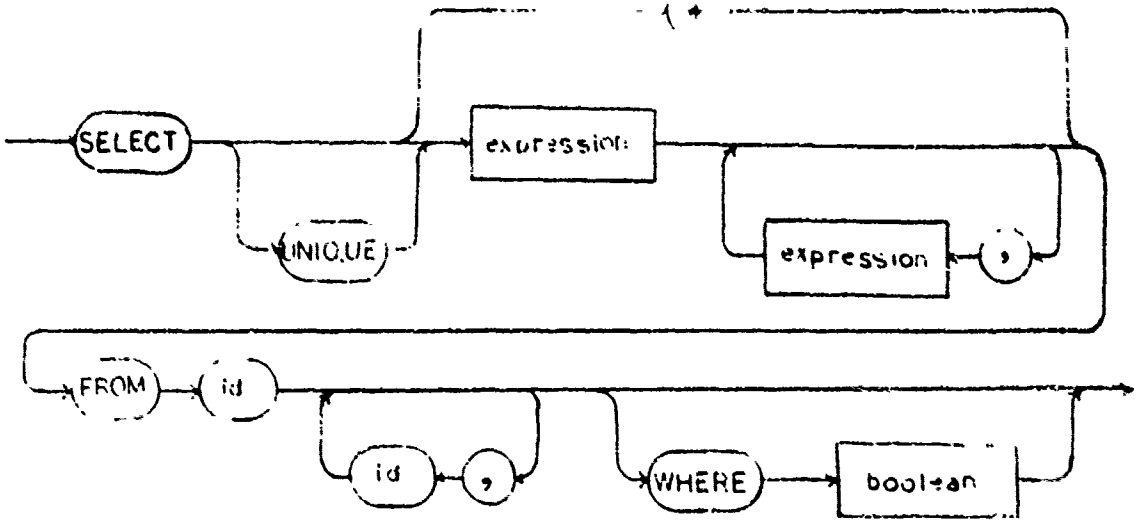


statement

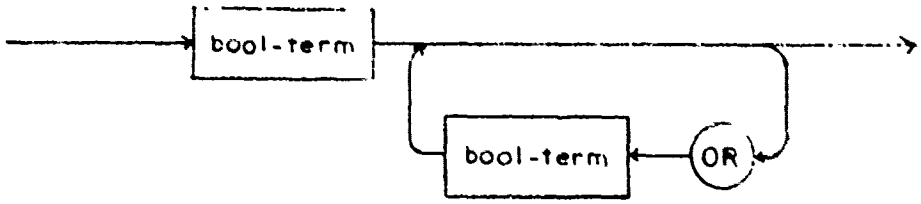


query-express

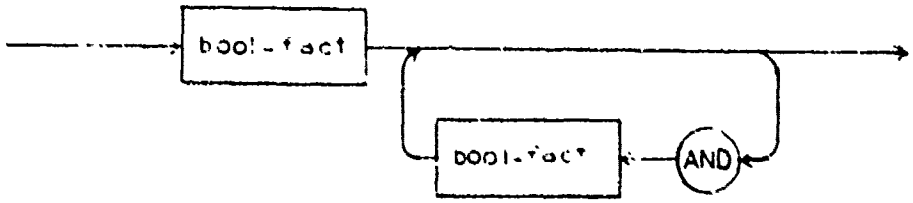




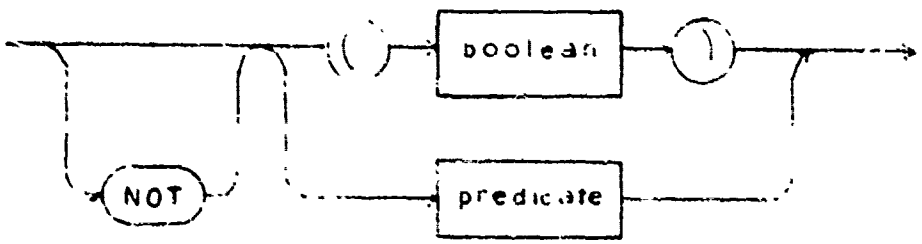
boolean



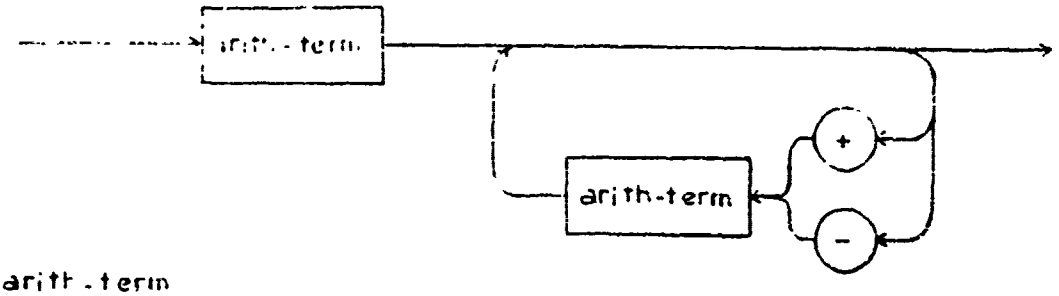
bool-term



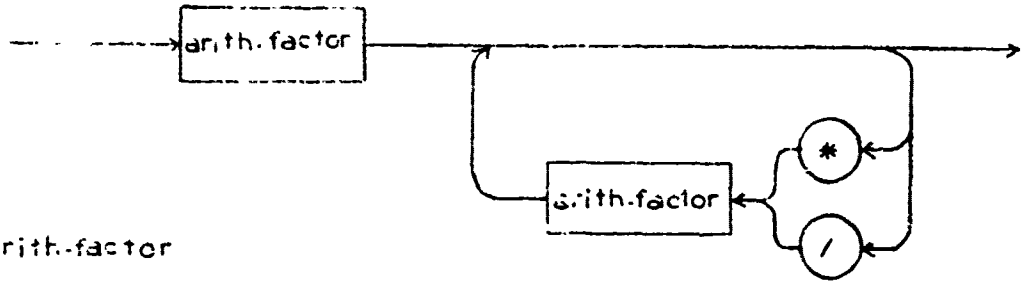
bool-fact



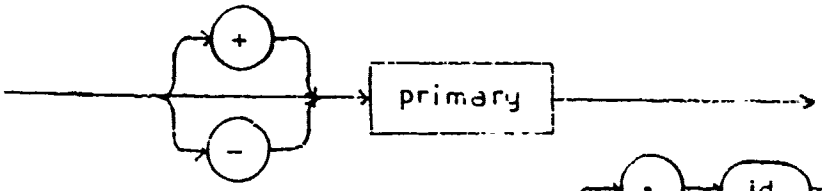
expression



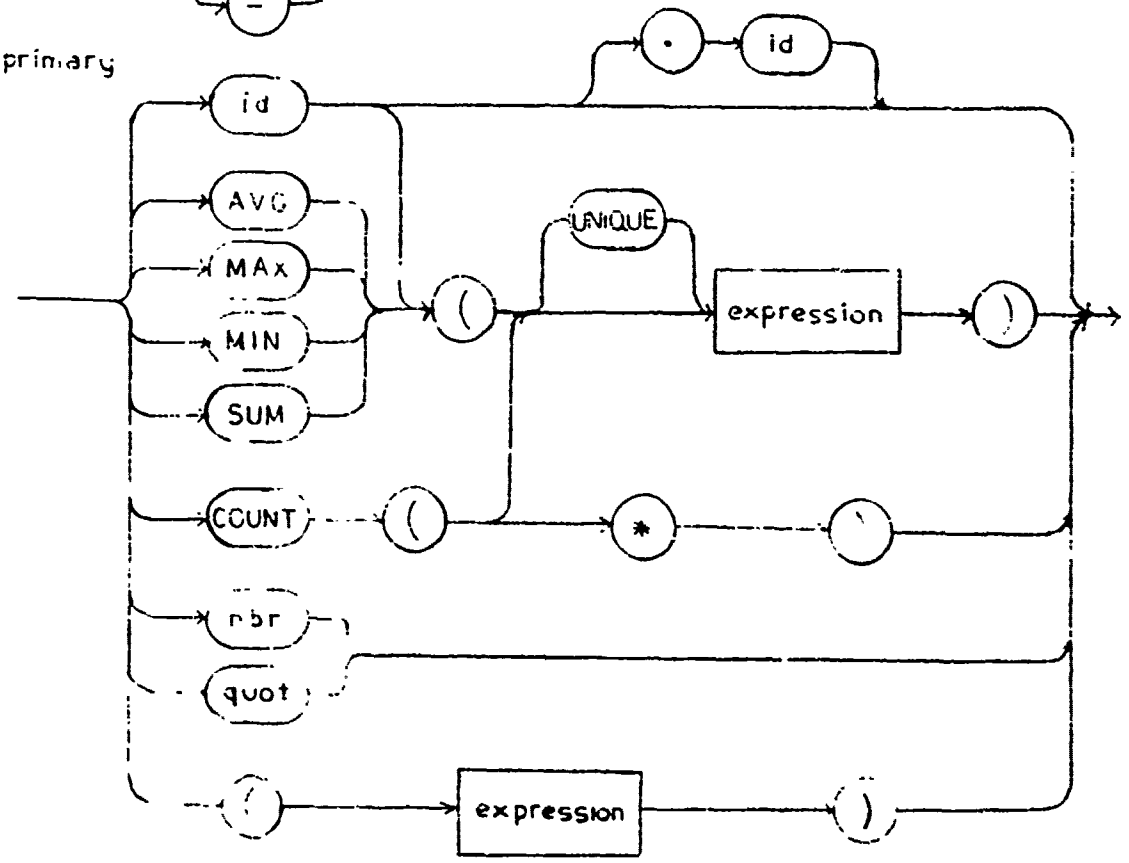
arith-term



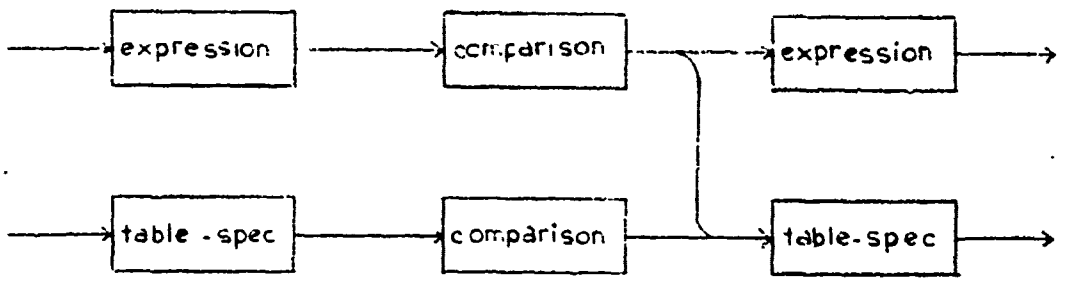
arith-factor



primary



predicate



comparison

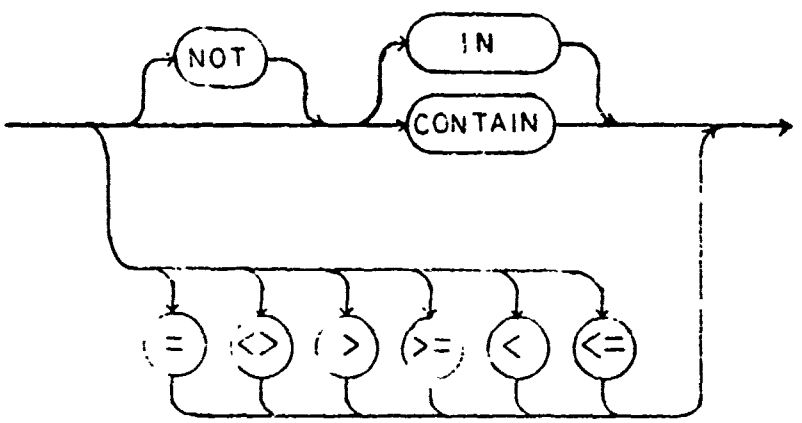
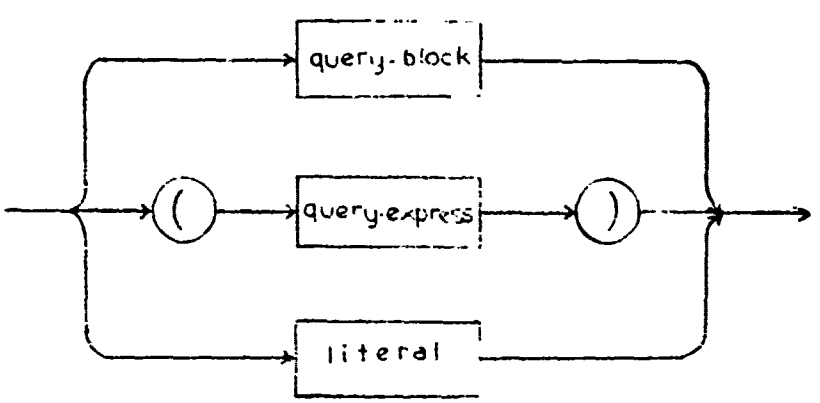
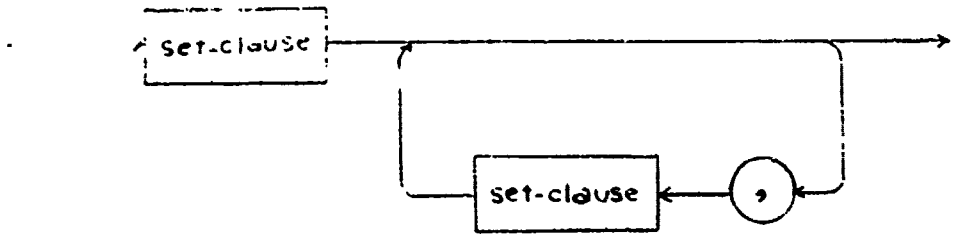


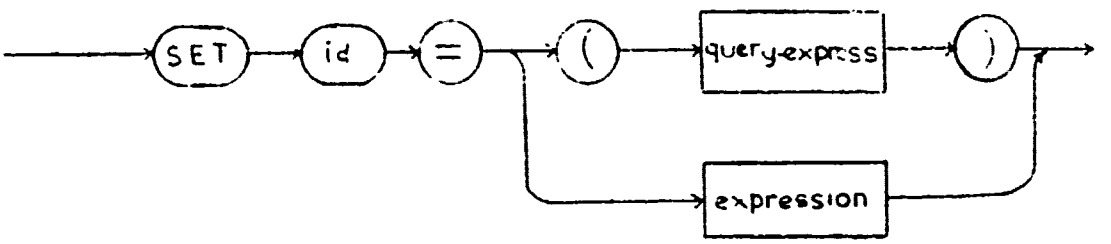
table-spec



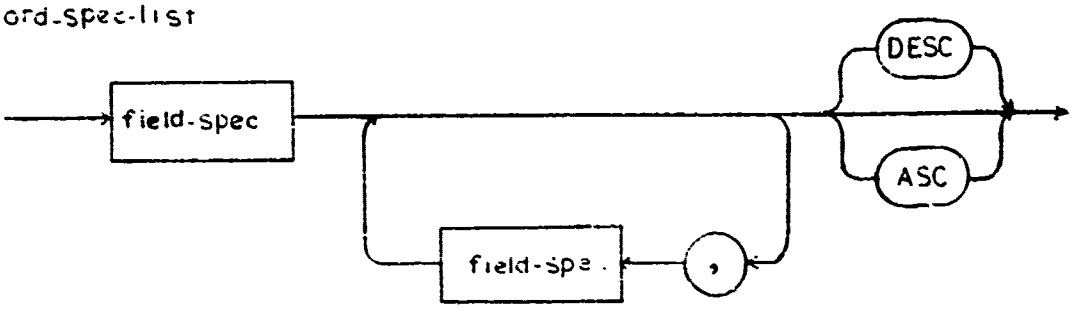
set-clause-list



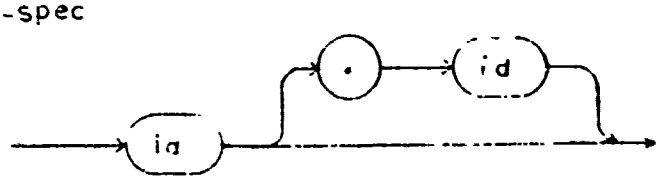
set-clause



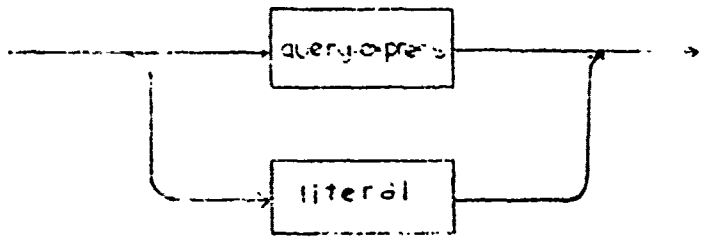
ord-spec-list



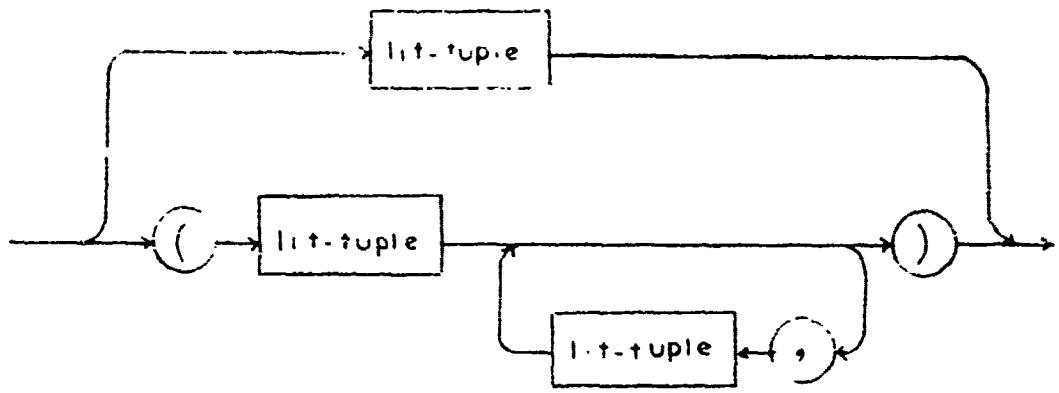
field-spec



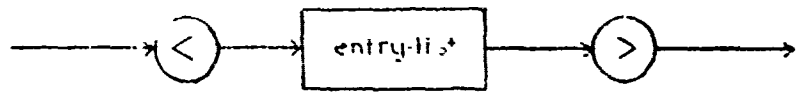
insert-spec



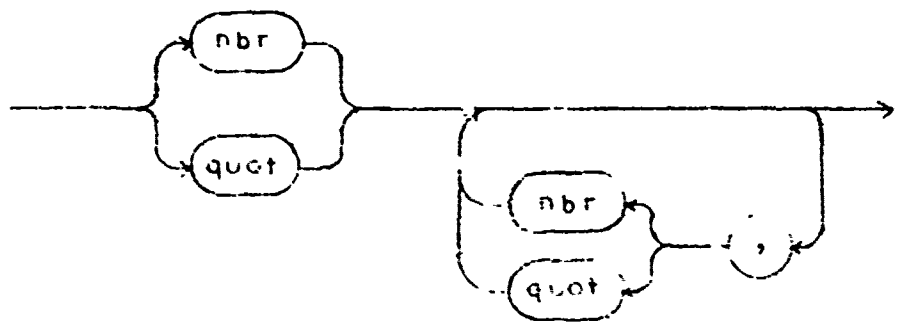
literal



lit-tuple



entry-list



```
(*****************************************************************)
(*                                                               *)
(*    THIS UNIT IS INTENDED TO CREATE INITIAL DATA BASE.         *)
(*    THE USER IS ASKED TO GIVE THE FILE NAME AND THE DATA       *)
(*              TYPE (SEE USER'S MANUAL)                         *)
(*                                                               *)
(*****************************************************************)

UNIT CREATE;

  INTERFACE

    CONST  MAXT     = 50; (* MAXIMUM NUMBER OF TUPLES  *)
                          (*    IN ONE RELATION        *)
    TYPE   TNBR     = 1..MAXT;
           INNON    = RECORD                 (*  RECORD WICH CONSISIS   *)
                        IDATA : INTEGER;     (*   OF ONE INTEGER KEY    *)
                        NONI  : STRING       (* AND ONE NON-RETRIEVABLE *)
                      END;                   (*     ATTRIBUTES          *)

           STNON    = RECORD                 (*  RECORD WHICH CONSISTS  *)
                        SDATA.               (*    OF ONE STRING KEY   *)
                        NONS  : STRING       (* AND ONE NON-RETRIEVABLE *)
                      END;                   (*     ATTRIBUTES          *)

           SETREC   = RECORD                      (* RECORD OF VARIABLE  *)
                        SETDATA : SET OF TNBR (* OF SET OF INTEGERS  *)
                      END;

    VAR    INTFILE : FILE OF INTEGER;
           STRFILE : FILE OF STRING;
           SETFILE : FILE OF SETREC;
           INON    : FILE OF INNON;
           SNON    : FILE OF STNON;

    PROCEDURE CREATES;

    IMPLEMENTATION
       VAR REC,
           NUM  : INTEGER;
           CH   : CHAR;
           INPUT,
           FNAME: STRING;

   PROCEDURE CREATES;

     PROCEDURE ENTERINT(F : STRING); (* CREATE FILE OF INTEGER *)
         BEGIN
           REWRITE('INTFILE.F');
           REC := 0;
           WHILE NOT EOF(INPUT) DO
             BEGIN
```

```
            SEEK(INTFILE,REC); GET(INTFILE);
            WRITE('=>');
            READLN(INTFILE^);
            IF NOT EOF(INPUT) THEN
              BEGIN
                SEEK(INTFILE,REC); PUT(INTFILE);
                REC := REC+1
              END
          END;
        CLOSE(INTFILE,LOCK)
      END;  (* ENTERDATA *)

  PROCEDURE ENTERSTR(F : STRING); (* CREATE FILE OF STRING *)
    BEGIN
      REWRITE(STRFILE,F);
      REC := 0;
      WHILE NOT EOF(INPUT) DO
        BEGIN
          SEEK(STRFILE,REC); GET(STRFILE);
          WRITE('=>'); READLN(STRFILE^);
          IF NOT EOF(INPUT) THEN
            BEGIN
              SEEK(STRFILE,REC); PUT(STRFILE);
              REC := REC+1
            END;
        END;
      CLOSE(STRFILE,LOCK)
  END;

PROCEDURE ENTERSET(F : STRING);  (* CREATE FILE OF SET OF *)
  BEGIN                          (*       INTEGERS        *)
    REWRITE(SETFILE,F);
    REC := 0;
    WHILE NOT EOF(INPUT) DO
      BEGIN
        SEEK(SETFILE,REC); GET(SETFILE);
        WRITE('=>');
        WITH SETFILE^ DO
          BEGIN
            SETDATA := [];
            REPEAT
              READ(NUM); SETDATA := SETDATA + [NUM]
            UNTIL EOLN(INPUT)
          END;
        IF NOT EOF(INPUT) THEN
          BEGIN
            SEEK(SETFILE,REC); PUT(SETFILE);
            REC := REC+1
          END;
      END;
    CLOSE(SETFILE,LOCK)
  END;
```

```
PROCEDURE INTANDNON(F : STRING);  (* CREATE FILE OF RECORDS *)
  BEGIN                           (*  WICH CONSISTS OF ONE  *)
    REWRITE(INON,F);              (*  INTEGER AND ONE NON-  *)
    REC := 0;                     (* RETRIEVABLE ATTRIBUTES *)
    WHILE NOT EOF(INPUT) DO
      BEGIN
        SEEK(INON,REC); GET(INON); WRITE('=>');
        WITH INON^ DO
          BEGIN READLN(IDATA);
            WRITE('=>'); READLN(NONI)
          END;
        IF NOT EOF(INPUT) THEN
          BEGIN SEEK(INON,REC); PUT(INON);
            REC := REC+1
          END
      END;
    CLOSE(INON,LOCK)
  END;

PROCEDURE STRANDNON(F : STRING); (* CREATE FILE OF RECORDS *)
  BEGIN                          (*  WICH CONSISTS OF ONE  *)
    REWRITE(SNON,F);             (*   STRING AND ONE NON-  *)
    REC := 0;                    (* RETRIEVABLE ATTRIBUTES *)
    WHILE NOT EOF(INPUT) DO
      BEGIN
        SEEK(SNON,REC); GET(SNON); WRITE('=>');
        WITH SNON^ DO
          BEGIN
            READLN(SDATA);
            WRITE('=>'); READLN(NONS)
          END;
        IF NOT EOF(INPUT) THEN
          BEGIN SEEK(SNON,REC); PUT(SNON);
            REC := REC+1
          END
      END;
    CLOSE(SNON,LOCK)
  END;


  BEGIN  (* CREATES *)
    WRITELN;
    WRITELN('The system is ready to create a file');
    WRITE('Please enter your file name -------->');
    READLN(INBUF);
    ENAME := CONCAT('#5:',INBUF);
    WRITE('Any non-retrieval attribute entered here? Y/N  ->');
    READ(CH); WRITELN;
    IF CH = 'Y' THEN
```

```
        WRITE('Please give its name ---------------->');
      WRITELN; WRITE('Data type? INT(eger, STR(ing, SET. ->');
      READLN(INBUF);
      WRITELN('You may enter your data');
        CASE CE OF
          'N' : IF INBUF = 'INT' THEN ENTERINT(FNAME)
                  ELSE IF INBUF = 'STR' THEN ENTERSTR(FNAME)
                    ELSE IF INBUF = 'SET' THEN ENTERSET(FNAME)
                      ELSE BEGIN
                        WRITELN('Fail to create a file');
                        EXIT(CREATES)
                      END;
          'Y' : IF INBUF = 'INT' THEN INTANDNON(FNAME)
                  ELSE IF INBUF = 'STR' THEN STRANDNON(FNAME)
                    ELSE BEGIN
                      WRITELN('Fail to create a file');
                      EXIT(CREATES)
                    END
        END;
      WRITELN('File with name ',FNAME,' has been created');
      WRITELN('with ',REC:4,' records')
  END;  (* CREATES *)
END.  (* UNIT CREATE *)
```

```
(*****************************************************)
(*                                                   *)
(*       THIS UNIT IS INTENDED TO GIVE THE USER      *)
(*    INFORMATION ABOUT THE EXISTING DATA BASE AND   *)
(*              HOW TO USE THE SYSTEM                *)
(*                                                   *)
(*****************************************************)

UNIT HELPS;

  INTERFACE

    VAR  TXT : TEXT;

    PROCEDURE    HELP;

  IMPLEMENTATION

    VAR I, RES : INTEGER;
        CH     : CHAR;
        FNAME  : STRING;

    PROCEDURE HELP;

      PROCEDURE READTXT(S : STRING);

        BEGIN
          RESET(TXT,S); I := 0;
            REPEAT
              WHILE NOT EOLN(TXT) DO
                BEGIN READ(TXT,CH); WRITE(CH) END;
              READLN(TXT); I := I+1;
              WRITELN;
            UNTIL (I = 20) OR ( EOF(TXT));
            IF NOT EOF(TXT) THEN
              BEGIN READ(CH);
                IF CH = ' ' THEN I := 0 END;
          CLOSE(TXT,LOCK)
        END;

      BEGIN  (* HELP *)
        WRITELN; WRITE('D(ata Base, U(ser Manual ->');
        READ(CH); WRITELN;
        CASE CH OF
          'D' : FNAME := 'DATABASE';  (* BOTH FILES ARE STORED  *)
          'U' : FNAME := 'USERMANUAL' (*     IN SYSTEM DISK     *)
        END;
        READTXT(FNAME)
      END;
  END.  (* UNIT HELPS *)
```

```
{*************************************************}
{*                                               *}
{*  THIS UNIT IS INTENDED TO VERIFY USER'S REQUEST *}
{*     WRITTEN IN SEQUEL-LIKE QUERY LANGUAGE.    *}
{*  ERROR MESSAGE IS GIVEN ON EACH SYNTAX ERROR  *}
{*  AND THE USER IS ASKED TO REPEAT. I/O ERROR,  *}
{*  HOWEVER, IS HANDLED BY UCSD-PASCAL SYSTEM. IF *}
{*  IT HAPPENS, THE SYSTEM QUITS AND HAS TO BE RE- *}
{*                INITIALIZED.                   *}
{*  BEFORE ENTERING THE REQUEST, THE USER IS ASKED *}
{*  TO DEFINE HIS OR HER DATA BASE WHICH HE OR SHE *}
{*  WANTS TO WORK ON. BY DOING THIS, ALL THE IN-   *}
{*  VERTED RELATIONSHIPS FILES CAN BE LOADED IN THE *}
{*     MAIN MEMORY TO OBTAIN FASTER ACCESS.       *}
{*                                               *}
{*************************************************}

UNIT PARSE;

INTERFACE

CONST  MAXATT  =  6;  (* MAXIMUM NUMBER OF ATTRIBUTES IN A *)
                      (*          RELATION                 *)
       MAXREL  =  3;  (* MAXIMUM NUMBER OF RELATIONS IN A  *)
                      (*          DATA  BASE               *)
       MAXFLD  = 15;  (* MAXIMUM NUMBER OF ALL ATTRIBUTES  *)
       MAXREF  = 10;  (* MAXIMUM NUMBER OF DATA ACCESSES   *)
       MAXTPL  = 50;  (* MAX. NUMBER OF TUPLES IN A RELATION *)
       MAXSYMS = 15;  (* MAX. NUMBER OF OPERATION SEQUENCE *)
       MAXINV  = 100; (* MAX. TOTAL NUMBER OF RETRIEVABLE  *)
                      (* DATA ITEMS,i.e. TOTAL NUMBER OF   *)
                      (*   INVERTED FILES' RECORDS         *)

TYPE   SYMBOL  = (NUL, IDENT, NBR, PLUS, MIN, STAR, SLASH,
                  EQL, NEQ, LSS, LEQ, GTR, GEQ, LPRN, RPRN,
                  CMA, CLN, PRD, QUOT, SELS, SELSYM, INSSYM,
                  INTOSYM, UPDSYM, DELSYM, UNISYM, WHERSYM,
                  ORSYM, ANDSYM, NOTSYM, CONTSYM, INSYM,
                  INTRSYM, UNISYM, MINSSYM, AVGSYM, MAXSYM,
                  MINSYM, SUMSYM, CNTSYM, ORDSYM, BYSYM,
                  ASCSYM, DESSYM, SETSYM, FRMSYM, NOCONT,
                  NOTIN,STSYM, ENDSYM, COND, LOD, RELSYM,
                  ATTSYM);
       TPLNO   = 1..MAXTPL;
       FIELDNO = 1..MAXATT;
       SETTUP  = SET OF TPLNO;
       STATUS  = (KEY, CKEY, RETR, NONRETR);
                 (* KEY, COMBINED KEY, RETRIEVABLE AND *)
                 (*        NON - RETRIEVABLE           *)
       RECREC  = RECORD          (* RECORD OF REFERENCE *)
```

```
                    RNO,                    (* RELATION NO.   *)
                    ANO    : INTEGER;       (* ATTRIBUTE NO.  *)
                    FCT    : SYMBOL;        (*    FUNCTION    *)
                    CASE SYMBOL OF          (*    VALUE       *)
                      SETS : (NAMES : SET OF FIELDNO);
                      QUOT : (QUOTS : STRING);
                      NBR  : (VAL   : INTEGER)
                  END;
  TRLREL  = RECORD          (* INFORMATION OF RELATION *)
                    NAMED  : STRING;  (*  RELATION NAME   *)
                    BASE   : INTEGER; (* ITS ATTRIBUTE'S  *)
                                      (* BASE LOCATION IN *)
                                      (* ATTRIBUTE TABLE  *)
                    SIZE   : INTEGER  (* NUMBER OF ATT    *)
                  END;
  ATT     = RECORD          (* INFORMATION OF ATTRIBUTE. *)
                    ATN    : STRING;  (*   ATTRIBUTE NAME   *)
                    STAT   : STATUS;
                    KIND   : SYMBOL;  (* STRING OR INTEGER  *)
                    ADDR,             (* BASE ADDRESS AFTER *)
                                      (* BEING LOADED INTO  *)
                                      (*  MAIN MEMORY       *)
                    SIZ    : INTEGER  (* NUMBER OF RECORDS  *)
                                      (*   LOADED           *)
                  END;
  INVREC  = RECORD                    (* SET OF TUPLE NO.   *)
                    SETNO  : SETTUP   (* OF ASSOCIATED DATA *)
                  END;                (* ITEM               *)

VAR   STM, FUNC, OBJ : SYMBOL;
      A, B, C,CX, AA,ZZP,
      I, J, K, P, TT,TOP,
      CC, EE, LL , NUM : INTEGER;
      X       : INTERACTIVE;
      QUOTES,QSTR,
      STR,STRG,
      FNAME, RELNAME : STRING;
      ATTS    : SET OF FIELDNO;
      ST      : STATUS;
      RELTBL  : ARRAY[1..MAXTRL] OF TRLREL; (* RELATION LIST *)
      ATTBL   : ARRAY[1..MAXFLD] OF ATT;    (* ATTRIBUTE LIST *)
      SEQTBL  : ARRAY[1..MAXSYMS] OF SYMBOL; (* SEQUENCE      *)
      INVTBL  : ARRAY[1..MAXINV] OF SETTUP;  (* TABLE OF SET *)
                                             (* OF TUPLE NO. *)
      TEMP    : ARRAY[1..MAXATT] OF STRING;
      INS     : REFREC;
      REFTBL  : ARRAY[1..MAXREF] OF REFREC;  (* LIST OF REF. *)
      ANYREL  : FILE OF INVERT;
      ALL,                            (* ALL ATTRIBUTES NEEDED *)
      NOTER,                          (*    NO ERRORS          *)
      NOCOND  : BOOLEAN;              (*  NO QUALIFICATIONS    *)
```

```
PROCEDURE INITIALIZE;
PROCEDURE PREPARE;
PROCEDURE ERROR(S : SYMBOL);
PROCEDURE SCANNER;
PROCEDURE STATEMENT;

IMPLEMENTETION

CONST   WRDSIZE = 10;  (*    MAXIMUM SIZE OF IDENTIFIER      *)
        NORW    = 26;  (*    NUMBER OF RESERVED WORDS IN     *)
                       (*        QUERY LANGUAGE              *)
        NMAX    = 10;
TYPE    STRG10  = PACKED ARRAY[1..WRDSIZE] OF CHAR;
VAR     CH      : CHAR;
        REC     : INTEGER;
        WORD.ID : STRG10;
        RWD     : ARRAY[1..NORW] OF STRG10;
        WSYM    : ARRAY[1..NORW] OF SYMBOL;
        SSYM    : ARRAY[CHAR] OF SYMBOL;
        LINE    : PACKED ARRAY[1..80] OF CHAR;

 (*****************************************************************)
 (*                                                               *)
 (* INITIALIZE THE SYSTEM WITH RESERVED WORDS', SYMBOL'S,         *)
 (* SPECIAL CHARACTERS' AND SOME INITIAL VALUE DEFINITIONS        *)
 (*                                                               *)
 (*****************************************************************)

PROCEDURE INITIALIZE;
  BEGIN (* INITIALIZE *)
    RWD[1]  := 'AND       ';   WSYM[1]  := ANDSYM;
    RWD[2]  := 'ASC       ';   WSYM[2]  := ASCSYM;
    RWD[3]  := 'AVG       ';   WSYM[3]  := AVGSYM;
    RWD[4]  := 'BY        ';   WSYM[4]  := BYSYM;
    RWD[5]  := 'CONTAIN   ';   WSYM[5]  := CONTSYM;
    RWD[6]  := 'COUNT     ';   WSYM[6]  := CNTSYM;
    RWD[7]  := 'DELETE    ';   WSYM[7]  := DELSYM;
    RWD[8]  := 'DESC      ';   WSYM[8]  := DESSYM;
    RWD[9]  := 'FROM      ';   WSYM[9]  := FRMSYM;
    RWD[10] := 'IN        ';   WSYM[10] := INSYM;
    RWD[11] := 'INSERT    ';   WSYM[11] := INSSYM;
    RWD[12] := 'INTERSECT ';   WSYM[12] := INTRSYM;
    RWD[13] := 'INTO      ';   WSYM[13] := INTOSYM;
    RWD[14] := 'MAX       ';   WSYM[14] := MAXSYM;
    RWD[15] := 'MIN       ';   WSYM[15] := MINSYM;
    RWD[16] := 'MINUS     ';   WSYM[16] := MINSSYM;
    RWD[17] := 'NOT       ';   WSYM[17] := NOTSYM;
    RWD[18] := 'OR        ';   WSYM[18] := ORSYM;
    RWD[19] := 'ORDER     ';   WSYM[19] := ORDSYM;
    RWD[20] := 'SELECT    ';   WSYM[20] := SELSYM;
    RWD[21] := 'SET       ';   WSYM[21] := SETSYM;
    RWD[22] := 'SUM       ';   WSYM[22] := SUMSYM;
```

```
    RWD[23] := 'UNION      ';     WSYM[23] := UNISYM;
    RWD[24] := 'UNIQUE     ';     WSYM[24] := UNIQSYM;
    RWD[25] := 'UPDATE     ';     WSYM[25] := UPDSYM;
    RWD[26] := 'WHERE      ';     WSYM[26] := WHERESYM;
    SSYM['+'] := PLUS;            SSYM['-'] := MIN;
    SSYM['*'] := STAR;            SSYM['/'] := SLASH;
    SSYM['='] := EQL;             SSYM['<'] := LSS;
    SSYM['>'] := GTR;             SSYM[':'] := CLN;
    SSYM[','] := CMA;             SSYM['.'] := PRD;
    SSYM['('] := LPRN;            SSYM[')'] := RPRN;
    CC := 0; LL := 0; CH := ' '
  END;  (* INITIALIZE *)

(*****************************************************************)
(*                                                               *)
(* ERROR MESSAGES ARE GIVEN ON SYNTAX ERROR AND DATA MANI-       *)
(* PULATION ERROR. UCSD - PASCAL I/O ERROR WILL QUIT THE         *)
(*         SYSTEM AND HAS TO BE REINITIALIZED.                   *)
(*                                                               *)
(*****************************************************************)

 PROCEDURE ERROR;
   BEGIN (* ERROR *)
     CASE S OF
       CLN      : WRITELN('":" is expected      ');
       PRD      : WRITELN('"." is expected      ');
       IDENT    : WRITELN('Identifier is needed ');
       INTOSYM  : WRITELN('"INTO" is expected   ');
       SETSYM   : WRITELN('"SET" is expected    ');
       BYSYM    : WRITELN('"BY" is expected     ');
       EQL      : WRITELN('"=" is expected      ');
       GTR      : WRITELN('">" is expected      ');
       FRMSYM   : WRITELN('"FROM" is expected   ');
       INSYM    : WRITELN('Comparison is expected');
       LPRN     : WRITELN('"(" is expected      ');
       RPRN     : WRITELN('")" is expected      ');
       QUOT     : WRITELN('This string is too long');
       NBR      : WRITELN('Number or quoted string?');
       NUL      : WRITELN('Too long for a number')
     END;
     CC := LL; SYM := NUL; NOERR := FALSE;
     CASE ERR OF
       1 : EXIT(PREPARE);
       2 : EXIT(STATEMENT)
     END
   END;
```

```
(*****************************************************************)
(*                                                               *)
(*  AFTER PUTTING THE USER'S REQUEST IN A LINE BUFFER,           *)
(*     THE SCANNER WILL SCAN IT TOKEN BY TOKEN.                  *)
(*                                                               *)
(*****************************************************************)

PROCEDURE SCANNER;
VAR I, J, K: INTEGER;
    INBUF  : STRING;

  PROCEDURE GETCH;

    BEGIN (* GETCH *)
      IF CC = LL THEN
        BEGIN
          CC := 0; LL := 0;
          WRITE('===> '); READLN(X,INBUF);
          IF NOT EOF(X) THEN
            BEGIN
              LL := LENGTH(INBUF);
              FOR I := 1 TO LL DO
                LINE[I] := INBUF[I];
              LL := LL+1;
              FOR I := LL TO 80 DO LINE[I] := ' '
            END ELSE EXIT(SCANNER)
        END;
      CC := CC+1; CH := LINE[CC]
    END;

  BEGIN (* SCANNER *)
    GETCH: WHILE CH = ' ' DO GETCH;
    IF CH = '''' THEN
    BEGIN (* QUOTED STRING *)
      K := 0;
      GETCH;
      WHILE (CH <> '''') AND (K<30) DO
      BEGIN
        K := K+1; QUOTED[K] := CH; GETCH
      END;
      IF CH = '''' THEN
        BEGIN SYM := QUOT; QSTR := COPY(QUOTED,1,K) END
      ELSE ERROR(QUOT)
    END ELSE
      BEGIN
        IF CH IN ['A'..'Z'] THEN
        BEGIN
          K := 0; KK := WRDSIZE;
          REPEAT
            IF K < WRDSIZE THEN
            BEGIN
              K := K+1; WORD[K] := CH
```

```
        END;
        GETCH
      UNTIL NOT (CH IN['A'..'Z','0'..'9','#']);
      CC := CC-1;
      IF K > KK THEN KK := K
      ELSE REPEAT
        WORD[KK] := ' '; KK := KK-1
        UNTIL KK =K;
      ID := WORD; I := 1; J := NORW;
      REPEAT
        K := (I+J) DIV 2;
        IF ID <= RWD[K] THEN J := K-1;
        IF ID >= RWD[K] THEN I := K+1
      UNTIL I > J;
      IF I-1 > J THEN SYM := WSYM[K]
      ELSE BEGIN SYM := IDENT;
        FOR I := 1 TO KK DO STRG[I] := ID[I];
        STR := COPY(STRG,1,KK) END;
    END ELSE
      IF CH IN ['0'..'9'] THEN
      BEGIN (* NUMBER *)
        K := 0; NUM := 0; SYM := NUM;
        REPEAT
          NUM := 10*NUM + (ORD(CH)-ORD('0'));
          K := K+1; GETCH
        UNTIL NOT (CH IN ['0'..'9']);
        CC := CC-1;
        IF K > NMAX THEN ERROR(NUL)
      END ELSE
        BEGIN
          IF CH = '<' THEN
          BEGIN GETCH;
            IF CH = '=' THEN SYM := LEQ
            ELSE IF CH = '>' THEN SYM := NEQ
              ELSE BEGIN CC := CC-1; SYM := LSS END
          END ELSE
            IF CH = '>' THEN
            BEGIN GETCH;
              IF CH = '=' THEN SYM := GEQ
              ELSE BEGIN CC := CC-1; SYM := GTR END
            END ELSE SYM := SSYM[CH]
        END
    END
END;  (* SCANNER *)
```

```
(*****************************************************)
(*                                                   *)
(*  THE NEXT TWO FUNCTIONS WILL IDENTIFY THE RELATION'S   *)
(*  NO. AND ATTRIBUTE'S NO.                          *)
(*                                                   *)
(*****************************************************)

 FUNCTION POSREL(A : STRING) : INTEGER;
   VAR  I : INTEGER;
   BEGIN
     I := 0;
     REPEAT I := I+1
     UNTIL (I = MAXREL) OR (RELTBL[I].NAMED = A);
     IF I <> MAXREL THEN POSREL := I
       ELSE ERROR(RELSYM)
   END;

 FUNCTION POSATT(VAR P : INTEGER; A : STRING) : INTEGER;
   BEGIN
     REPEAT P := P+1
     UNTIL (ATTBL[P].ATN = A) OR (P > P+C);
     IF NOT (P > B+C) THEN POSATT := P
       ELSE ERROR(ATTSYM)
   END;

(*****************************************************)
(*                                                   *)
(*  THIS PROCEDURE IS USED TO LOAD THE RELATIONSHIPS   *)
(*  INVERTED FILES INTO MAIN MEMORY. SINCE THE RELA-   *)
(*  TIONSHIPS CONTAIN NUMBERS ONLY, THIS WILL OCCUPY   *)
(*  LESS MEMORY, MUCH LESS THAN THE ASSOCIATED DATA.   *)
(*      PROCEDURES ARE AVAILABLE IN USER'S MANUAL      *)
(*                                                   *)
(*****************************************************)

PROCEDURE PREPARE;
  VAR I , J, K : INTEGER;

  PROCEDURE REL;

    PROCEDURE ATTDESC;       (* ATTRIBUTE DESCRIPTIONS *)
      BEGIN
        IF SYM = IDENT THEN
          BEGIN
            WITH RELTBL[J] DO
              BEGIN B := BASE; C := SIZE END;
            P := B; K := POSATT(P,STR);
            SCANNER;
            IF SYM = CMA THEN SCANNER ELSE ERROR(CMA);
            IF STR = 'KEY' THEN
              WITH ATTBL[K] DO
                BEGIN STAT := KEY; ADDR := 0 END
```

```
            ELSE IF (STR = 'CKEY') OR (STR = 'RETR') THEN
              BEGIN
                WITH ATTBL[K] DO
                  BEGIN IF STR = 'CKEY' THEN STAT := CKEY
                    ELSE STAT := RETR;
                    ADDR := AA
                  END;
                FNAME := CONCAT('#5:',RELTBL[J].NAMED);
                RELNAME:=CONCAT(FNAME,'.',ATTBL[K].ATN);
                REC := 0;
                RESET(ANYREL,RELNAME);
                  WHILE NOT EOF(ANYREL) DO
                    BEGIN SEEK(ANYREL,REC); GET(ANYREL);
                      IF NOT EOF(ANYREL) THEN
                        BEGIN AA := AA+1;
                        WITH ANYREL^ DO INVTBL[AA] := SFTNO;
                          REC := REC+1
                        END
                    END;
                    WITH ATTBL[K] DO SIZ := AA - ADDR;
                  CLOSE(ANYREL,LOCK)
              END ELSE IF STR = 'NONRETR' THEN
                WITH ATTBL[K] DO
                  BEGIN STAT := NONRETR; ADDR := 0; SIZ := 0 END
                ELSE ERROR(STSYM);
            SCANNER;
            IF SYM = CMA THEN SCANNER ELSE ERROR(CMA);
            IF STR = 'CH' THEN ATTBL[K].KIND := QUOT
              ELSE IF STR = 'NBR' THEN ATTBL[K].KIND := NBR
                ELSE ERROR(KNDSYM);
            IF CC < LL-1 THEN SCANNER
              ELSE BEGIN WRITE('          ');
                SCANNER END
          END ELSE ERROR(IDENT)
      END;

    PROCEDURE IDENTIFIER;
      BEGIN
        IF SYM = IDENT THEN
          BEGIN I := I+1;
            ATTBL[I].ATN := STR;
            SCANNER
          END ELSE ERROR(IDENT)
      END;

    PROCEDURE RELDESC;        (* RELATION DESCRIPTIONS *)
      BEGIN
        IF SYM = IDENT THEN
          BEGIN
            J := J+1; RELTBL[J].NAMED := STR;
            RELTBL[J].BASE := I; SCANNER;
            IF SYM = LPRN THEN
```

```
            BEGIN SCANNER;
              IDENTIFIER;
              WHILE SYM = CMA DO
                BEGIN SCANNER; IDENTIFIER END;
              RELTBL[J].SIZE := I-RELTBL[J].BASE;
              IF SYM = RPRN THEN
                BEGIN WRITE('Attribute '); SCANNER
                END ELSE ERROR(RPRN)
            END ELSE ERROR('LPRN')
        END ELSE ERROR(IDENT)
    END;

  BEGIN   (* REL *)
    RELDESC;
    ATTDESC;
    WHILE SYM = IDENT DO ATTDESC
  END;

BEGIN   * PREPARE *)
  I := 0; J := 0; ERR := 1;
  AA := 0; NOERR := TRUE;
  WRITELN('Please describe your Data Base');
  WRITE('Relation  ');
  SCANNER; REL;
  WHILE SYM = CMA DO
    BEGIN WRITE('Relation  ');
      SCANNER; REL
    END;
  IF SYM <> PRD THEN ERROR('PRD')
  ELSE WRITELN
END;


(*****************************************************************)
(*                                                               *)
(*  VERIFICATION PROCESS TAKES PLACE IN THIS PROCEDURE  *)
(*  FOLLOWING THE QUERY LANGUAGE GRAMMER. AT THE SAME   *)
(*  TIME TWO TABLES ARE CREATED IN ORDER TO GIVE THE    *)
(*  INFORMATION TO INTERPRETER WHAT ACTION SHOULD BE    *)
(*  TAKEN. THEY ARE SEQUENCE TABLE AND REFERENCE TABLE. *)
(*                                                               *)
(*****************************************************************)

PROCEDURE STATEMENT;
  VAR  TREF, TSEQ : INTEGER;
       SAVSYM : SYMBOL;
       SAVREF : REFREC;
       SAVTBL : ARRAY[1..MAXREF] OF REFREC;
       SAVSEQ : ARRAY[1..MAXSYMS] OF SYMBOL;
```

```
(*****************************************************)
(*                                                   *)
(*  THE REFERENCE TABLE IS CREATED BY THIS PROCEDURE  *)
(*                                                   *)
(*****************************************************)

PROCEDURE ENTER(A,B,NUM : INTEGER; P,C : SYMBOL);
  BEGIN
    CX := CX+1;
    WITH REFTBL[CX] DO
      BEGIN
        RNO    := A;
        ANO    := B;
        PCT    := P;
        CASE C OF
          SETS : NAMES := ATTS;
          QUOT : QUOTS := OSTR;
          NBR  : VAL   := NUM
        END
      END;
  END;  (* ENTER *)

(*****************************************************)
(*                                                   *)
(*      THE FOLLOWING PROCEDURE GENERATES THE        *)
(*              SEQUENCE  TABLE                      *)
(*                                                   *)
(*****************************************************)

  PROCEDURE GEN'S : SYMBOL');
    BEGIN
      TOP := TOP-1;
      SEQTBL[TOP] := S
    END;


PROCEDURE EXPRESS;                  (*    EXPRESSION     *)
  VAR  SAVSYM : SYMBOL;

  PROCEDURE ARITHTERM;              (*  ARITHMATIC TERM  *)
    VAR  SAVSYM : SYMBOL;

    PROCEDURE ARITHFACT;            (* ARITHMATIC FACTOR *)
      VAR  SAVSYM : SYMBOL;

      PROCEDURE PRIMARY;            (*    PRIMARY        *)

        PROCEDURE UNIC;
          BEGIN (* UNIC *)
            IF SYM = UNIOSYM THEN SCANNER;
            EXPRESS;
            IF SYM = RPRN THEN SCANNER
```

```
            ELSE ERROR(RPRN)
          END;  (* UNIQ *)

        BEGIN (* PRIMARY *)
          IF SYM IN [AVGSYM, MAXSYM, MINSYM, SUMSYM] THEN
          BEGIN FUNC := SYM; SCANNER;
            IF SYM = LPRN THEN
            BEGIN SCANNER; UNIQ
            END ELSE ERROR(LPRN)
          END ELSE
            IF SYM = IDENT THEN
            BEGIN SCANNER;
              IF SYM = PRD THEN
              BEGIN IF CC < LL-1 THEN
                BEGIN SCANNER;
                  IF SYM = IDENT THEN SCANNER
                  ELSE ERROR(IDENT)
                END;
              END ELSE
                IF SYM = LPRN THEN
                BEGIN SCANNER; UNIQ
                END;
            END ELSE
              IF SYM = CNTSYM THEN
              BEGIN FUNC := SYM; SCANNER;
                IF SYM = LPRN THEN
                BEGIN SCANNER;
                  IF SYM = STAR THEN
                  BEGIN SCANNER;
                    IF SYM = RPRN THEN SCANNER
                    ELSE ERROR(RPRN)
                  END ELSE UNIQ
                END ELSE ERROR(LPRN)
              END ELSE
                IF SYM IN [QUOT, NBR] THEN SCANNER
                ELSE IF SYM = LPRN THEN
                  BEGIN SCANNER; EXPRESS;
                    IF SYM = RPRN THEN SCANNER
                    ELSE ERROR(RPRN)
                  END
      END;  (* PRIMARY *)

    BEGIN (* ARITFACT *)
      IF SYM IN [PLUS, MIN] THEN          (* UNARY '+' OR '-' *)
        BEGIN SAVSYM := SYM; SCANNER END;
      PRIMARY;
      IF SAVSYM IN [PLUS, MIN] THEN GEN(SAVSYM)
    END;  (* ARITFACT *)

    BEGIN (* ARITFACT *)
      ARITFACT;
      WHILE SYM IN [STAR, SLASH] DO
```

```
        BEGIN SAVSYM := SYM;
          SCANNER; ARITHFACT;
          GEN(SAVSYM)
        END
    END;  (* ARITHTERM *)

  BEGIN (* EXPRESS *)
    ARITHTERM;
    WHILE SYM IN [PLUS, MIN] DO
      BEGIN SAVSYM := SYM;
        SCANNER; ARITHTERM;
        GEN(SAVSYM)
      END
  END;  (* EXPRESS *)

PROCEDURE ENTRYLIST;              (*  ENTRY LIST *)

  PROCEDURE ENTRYLPRM;
    BEGIN (* ENTRYPRM *)
      IF SYM = CMA THEN
      BEGIN SCANNER;
        IF SYM IN [QUOT, NBR] THEN
        BEGIN SCANNER; ENTRYLPRM
        END ELSE ERROR(NBR)
      END
    END;  (* ENTRYLPRM *)

  BEGIN (* ENTRYLIST *)
    IF SYM IN [QUOT, NBR] THEN
      BEGIN SCANNER; ENTRYLPRM
      END ELSE ERROR(NBR)
  END;  (* ENTRYLIST *)

PROCEDURE LITTUPLE;               (*  LITERAL TUPLE  *)
  BEGIN (* LITTUPLE *)
    IF SYM = LSS THEN
      BEGIN SCANNER; ENTRYLIST;
        IF SYM = GTR THEN SCANNER
        ELSE ERROR(GTR)
      END ELSE ERROR(LSS)
  END;  (* LITTUPLE *)

PROCEDURE LITERAL;                (* LITERAL *)
  BEGIN
    LITTUPLE;
    WHILE SYM = CMA DO
      BEGIN
        SCANNER; LITTUPLE
      END;
    IF SYM = RPRN THEN SCANNER
    ELSE ERROR(RPRN)
  END;
```

```
PROCEDURE POOL; FORWARD;

PROCEDURE QUERYBLOCK; (* QUERY BLOCK WHERE SELECT-FROM-WHERE *)
                      (*       CLAUSE IS FOUND                *)
  VAR  SAVREF : REFREC;
       SAVSYM : SYMBOL;

  PROCEDURE FROMPRM;
    BEGIN (* FROMPRM *)
      IF SYM = CMA THEN
      BEGIN SCANNER;
        IF SYM = IDENT THEN
        BEGIN SCANNER; FROMPRM
        END ELSE ERROR(IDENT)
      END
    END;  (* FROMPRM *)

  BEGIN (* QUERYBLOCK *)
    IF SYM = SELSYM THEN
    BEGIN TT := 1; ATTS := []; ALL := FALSE;
      SCANNER;
      IF SYM = UNIQSYM THEN
        BEGIN SAVSYM := SYM; SCANNER END;
      IF SYM = STAR THEN BEGIN ALL := TRUE; SCANNER END
        ELSE BEGIN EXPRESS; TEMP[TT] := STR;
          WHILE SYM = CMA DO
            BEGIN TT := TT+1; SCANNER;
              EXPRESS; TEMP[TT] := STR
            END  END;
      IF SYM = FRMSYM THEN
      BEGIN SCANNER;
        IF SYM = IDENT THEN
        BEGIN
          A := POSREL(STR);              (*   RELATION NO.  *)
          P := RELTBL[A].BASE;           (* BASE OF ITS ATT *)
          C := RELTBL[A].SIZE;           (* # OF ATTRIBUTES *)
          IF ALL THEN     (* ALL ATT'S NEEDED IN INQUIRY *)
            FOR I := P+1 TO P+C DO ATTS := ATTS + [I]
          ELSE BEGIN
            P := P;
            FOR I := 1 TO TT DO
              ATTS := ATTS + [POSATT(P,TEMP[I])]
            END;
          ENTER(A,P,C,FUNC,SETS);
          SAVREF := REFTBL[CX]; CX := CX-1;
          SCANNER; FROMPRM;
          IF SYM = WHERSYM THEN
            BEGIN SCANNER; POOL;
              CX := CX+1; REFTBL[CX] := SAVREF;
              GEN(SELSYM)
```

```
            END ELSE NOCOND := TRUE; (* NO QUALIFICATIONS *)
          END ELSE ERROR(IDENT)
        END ELSE ERROR(FRMSYM)
      END ELSE ERROR(SELSYM)
    END;  (* QUERYBLOCK *)

  PROCEDURE QUERYEXPR;              (* QUERY EXPRESSION *)
    VAR  SAVSYM : SYMBOL;
    BEGIN (* QUERYEXPR *)
      IF SYM = LPRN THEN
      BEGIN SCANNER; QUERYEXPR;
        IF SYM = RPRN THEN SCANNER
        ELSE ERROR(RPRN)
      END ELSE QUERYBLOCK;
      WHILE SYM IN [INTRSYM, UNISYM, MINSYM] DO
        BEGIN
          SAVSYM := SYM; QUERYBLOCK;
          GEN(SAVSYM)
        END
    END;  (* QUERYEXPR *)

  PROCEDURE BOOL;        (* QUALIFICATION OR CONDITION *)

    PROCEDURE BOOLTERM;

    PROCEDURE PREDICATE;
      VAR SAVPTR : PTRREC;
          SAVSYM : SYMBOL;

      PROCEDURE TABLESPEC;
        BEGIN (* TABLESPEC *)
          IF SYM = SELSYM THEN QUERYBLOCK
            ELSE IF SYM = LSS THEN LITTUPLE
              ELSE IF SYM = LPRN THEN
                BEGIN SCANNER;
                  IF SYM = LSS THEN LITERAL
                  ELSE BEGIN
                    QUERYEXPR;
                    IF SYM = RPRN THEN SCANNER
                    ELSE ERROR(RPRN)
                  END
                END
        END;  (* TABLESPEC *)

      PROCEDURE COMPARISON;
        BEGIN (* COMPARISON *)
          IF SYM = NOTSYM THEN
            BEGIN SCANNER;
              IF SYM = CONTSYM THEN SYM := NOCONT
                ELSE IF SYM = INSYM THEN SYM := NOTIN
                  ELSE ERROR(NOTSYM);
              FUNC := SYM; SCANNER
```

```
        END ELSE
          IF SYM IN [EQL, NEQ, LSS, LEQ, GTR, GEQ,
                CONSTSYM, INSYM] THEN
            BEGIN FUNC := SYM; SCANNER END
          ELSE ERROR(INSYM)
      END;  (* COMPARISON *)

    BEGIN (* PREDICATE *)
      IF SYM IN [SELSYM,LPRN, LSS] THEN
      BEGIN TABLESPEC; COMPARISON; TABLESPEC
      END ELSE
        BEGIN EXPRESS;
          P := 3; I := POSART(P,STR);
          OBJ := ATTRL[I].FIND;
          COMPARISON;
          IF SYM IN [SELSYM, LPRN, LSS] THEN
            BEGIN OSTR := ' '; SAVSYM := LOD;
              ENTER(3,I,0,FUNC,OBJ);
              SAVREF := REFTBL[CX]; CX := CX-1;
              TABLESPEC;
              CX := CX+1; REFTBL[CX] := SAVREF
            END ELSE BEGIN EXPRESS;
              ENTER(3,I,VAR,FUNC,OBJ) END
        END;
      IF SAVSYM = LOD THEN GEN('LOD');
      GEN('CONJ')
    END;  (* PREDICATE *)

    PROCEDURE BOOLFACT;
      VAR   SAVSYM : SYMBOL;
      BEGIN (* BOOLFACT *)
        IF SYM = NOTSYM THEN
          BEGIN SAVSYM := SYM; SCANNER
          END ELSE SAVSYM := NUL;
        IF SYM = LPRN THEN
          BEGIN SCANNER; BOOL;
            IF SYM = RPRN THEN SCANNER
            ELSE ERROR(RPRN)
          END ELSE PREDICATE;
        IF SAVSYM = NOTSYM THEN GEN('NOTSYM')
      END;  (* BOOLFACT *)

      BEGIN (* BOOLTERM *)
        BOOLFACT;
        WHILE SYM = ANDSYM DO
          BEGIN SCANNER; BOOLFACT; GEN('ANDSYM') END
      END;  (* BOOLTERM *)

    BEGIN (* BOOL *)
      BOOLTERM;
      WHILE SYM = ORSYM DO
        BEGIN SCANNER; BOOLTERM; GEN('ORSYM') END
```

```
    END;  (* TOOL *)

  PROCEDURE OPDSPECLIST;        (* OPTER SPECIFICATION LIST *)

    PROCEDURE FIELDSPEC;
      BEGIN (* FIELDSPEC *)
        IF SYM = IDENT THEN
        BEGIN SCANNER;
          IF SYM = BRD THEN
          BEGIN IF CC < LL-1 THEN
            BEGIN SCANNER;
            IF SYM = IDENT THEN SCANNER
            ELSE ERROR(IDENT)
            END;
          END
        END ELSE ERROR(IDENT)
      END;  (* FIELDSPEC *)

    BEGIN (* OPDSPECLIST *)
      FIELDSPEC;
      WHILE SYM = CMA DO
        BEGIN SCANNER; FIELDSPEC END;
      IF SYM IN [ASCSYM, DESSYM] THEN SCANNER
    END;  (* OPDSPECLIST *)

  PROCEDURE INSERTSPEC;         (* INSERT SPECIFICATION *)
    BEGIN  * INSERTSPEC *)
      IF SYM = LSS THEN LITTUPLE
      ELSE IF SYM = SELSYM THEN
        BEGIN QUERYBLOCK;
          WHILE SYM IN [INTOSYM, UNISYM, MINSSYM] DO
            BEGIN SCANNER; QUERYBLOCK END;
        END ELSE
          IF SYM = LPRN THEN
            BEGIN SCANNER;
              IF SYM = LSS THEN LITERAL
              ELSE BEGIN
                QUERYEXPR;
                IF SYM = RPRN THEN
                  BEGIN SCANNER;
                    WHILE SYM IN [INTOSYM, UNISYM, MINSSYM] DO
                      BEGIN SCANNER; QUERYBLOCK END
                  END ELSE ERROR('RPRN')
                END
            END ELSE ERROR('LPRN')
    END;   * INSERTSPEC *)
```

```
PROCEDURE SETCLLIST;                    (* SET CLAUSE LIST *)
  VAR  SAVREF : REFREC;
       J      : INTEGER;

  PROCEDURE SETCLAUSE;
    BEGIN (* SETCLAUSE *)
      IF SYM = SETSYM THEN
      BEGIN SCANNER;
        IF SYM = IDENT THEN
        BEGIN P := B; I := POSATT(P,STR);
          OBJ := ATTBL[I].KIND; SCANNER;
          IF SYM = EQL THEN
          BEGIN FUNC := SYM; SCANNER;
            IF SYM = LPRN THEN
            BEGIN SCANNER; QUERYEXPR; GEN(LOD);
              IF SYM = RPRN THEN SCANNER
              ELSE ERROR(RPRN)
            END ELSE EXPRESS
          END ELSE ERROR(EQL)
        END ELSE ERROR(IDENT)
      END ELSE ERROR(SETSYM)
    END;  (* SETCLAUSE *)


  BEGIN (* SETCLLIST *)
    SETCLAUSE; ENTER(A,I,NUM,FUNC,OBJ);
    GEN(SETSYM);
    WHILE SYM = CMA DO
      BEGIN SCANNER; SETCLAUSE;
        ENTER(A,I,NUM,FUNC,OBJ);
        GEN(SETSYM)
      END;
    FOR J := 1 TO CX DO SAVTBL[J] := REFTBL[J];
      TREF := CX; CX := 0;
    FOR J := 1 TO TOP DO SAVSEQ[J] := SEQTBL[J];
      TSEQ := TOP; TOP := 0;
  END;  (* SETCLLIST *)


  BEGIN (* STATEMENT *)
    CX := 0; TOP := 0; ERR := 2;
    TREF := 0; TSEQ := 0;
    IF SYM = INSSYM THEN
    BEGIN                                   (* INSERTION *)
      SCANNER;
      IF SYM = INTOSYM THEN
      BEGIN SCANNER;
        IF SYM = IDENT THEN
        BEGIN SCANNER;
          IF SYM = CLN THEN
          BEGIN SCANNER; INSERTSPEC
          END ELSE ERROR(CLN)
```

```
      END ELSE ERROR(IDENT)
    END ELSE ERROR(INTOSYM)
  END ELSE
    IF SYM = UPDSYM THEN
    BEGIN                                (* UPDATE *)
      SCANNER;
      IF SYM = IDENT THEN
      BEGIN A := POSREL(STR); B := RELTBL[A].BASE;
        C := RELTBL[A].SIZE; SCANNER; SETCLLIST;
        IF SYM = WHERSYM THEN
        BEGIN SCANNER; BOOL;
          FOR I := 1 TO TREF DO
            BEGIN CX := CX+1;
              REFTBL[CX] := SAVTBL[I]
            END;
          FOR I := 1 TO TSEC DO
            BEGIN TOP := TOP+1;
              SECTBL[TOP] := SAVSEC[I]
            END
        END
      END ELSE ERROR(IDENT)
    END ELSE
      IF SYM = DELSYM THEN
      BEGIN                            (* DELETION *)
        SAVSYM := SYM; SCANNER;
        IF SYM = IDENT THEN
          BEGIN A := POSREL(STR); B := RELTBL[A].BASE;
            C := RELTBL[A].SIZE;
            ENTER(A,0,0,FUNC.NPR);
            SAVREF := REFTBL[CX]; CX := CX-1;
            SCANNER;
            IF SYM = WHERSYM THEN
            BEGIN SCANNER; BOOL;
              CX := CX+1; REFTBL[CX] := SAVREF;
              GEN(SAVSYM)
            END
          END ELSE ERROR(IDENT)
      END ELSE
        BEGIN                        (* INQUIRY *)
          QUERYEXPR;
          IF SYM = ORDSYM THEN
            BEGIN SCANNER;
              IF SYM = BYSYM THEN
                BEGIN SCANNER; ORDSPECLIST
                END ELSE ERROR(BYSYM)
            END;
        END
  END;  (* STATEMENT *)

END.    * UNIT PARSE *)
```

```
(*************************************************************)
(*                                                           *)
(*  THIS UNIT IS USED TO INTERPRET THE USER'S REQUEST        *)
(*  BY INTERPRETING THE TABLES AS THE RESULTS OF THE         *)
(*                PARSER ABOVE                               *)
(*                                                           *)
(*************************************************************)

UNIT INTERPRETER;

INTERFACE

  USES PARSE;

  TYPE     RELATION  = RECORD
                         NAME  : INTEGER;
                         DEPT  : INTEGER;
                         SKILL : INTEGER;
                         SAL   : INTEGER;
                         MANNO : INTEGER
                       END;
           CHDREC    = RECORD
                         NAME   : INTEGER;
                         CHNAME : INTEGER;
                         SEX    : INTEGER;
                         AGE    : INTEGER
                       END;

  VAR      L,M,N,
           PARENNO   : INTEGER;
           TUPLE     : RELATION;
           TABLE     : FILE OF RELATION;
           CHDTPL    : FILE OF CHDREC;
           DATA,
           ANYDATA,
           DEPTDATA,
           SALDATA,
           AGEDATA   : FILE OF INTEGER;
           STRDATA,
           NAMEDATA,
           SKILLDATA,
           MANNODATA,
           CHDDATA,
           SEXDATA : FILE OF STRING;

PROCEDURE INTRPRET;

  IMPLEMENTETION

    VAR   BLNK,
          INPUT,
```

```
            RNAME,
            DNAME     : STRING;
            SFO       : SYMBOL;
            SETNO,
            SETREC,
            SETTPL    : SETTUP;
            TEMPS     : ARRAY[1..6] OF SETTUP;
            NEXT      : BOOLEAN;

PROCEDURE INTRPRET;

{*****************************************************************}
{*                                                               *}
{*    THIS PROCEDURE WILL GIVE ALL THE TUPLE NUMBERS             *}
{*      WHICH THE CONDITION OR QUALIFICATION GIVEN               *}
{*                                                               *}
{*****************************************************************}

PROCEDURE CONDITION(RNO,ANO,VAL : INTEGER; STAT : STATUS;
                    FCT,KIND : SYMBOL; QUOTS : STRING);
  VAR   A, B,
        I, J : INTEGER;

  PROCEDURE GETTUPLENO(FUNC : SYMBOL; A, B : INTEGER);

    BEGIN
      SETTPL := [];
      FOR J := 1 TO B DO
        CASE FUNC OF
          INSYM : IF J IN SETREC THEN
                    SETTPL := SETTPL + INVTBL[A+J];
          NOTIN : IF NOT (J IN SETREC) THEN
                    SETTPL := SETTPL + INVTBL[A+J]
        END;
    END;  (* GETTUPLENO *)

  BEGIN
    WITH ATTPL[ANO] DO
      BEGIN
        DNAME := CONCAT('#5:',ATN);
        A := ADDR; B := SIZ
      END;
    IF NOT NEXT THEN
    BEGIN
      CASE KIND OF
        VBR : BEGIN
                RESET(DATA,DNAME);
                DARECNO := 0; SETREC := [ ];
                WHILE NOT EOF(DATA) DO
                BEGIN
                  CASE FCT OF
                    EQL : IF DATA^ = VAL THEN
```

```
                                SETREC := SETREC+[DARECNO+1];
                        NEQ : IF DATA^ <> VAL THEN
                                SETREC:= SETREC+[DARECNO+1];
                        LSS : IF DATA^ < VAL THEN
                                SETREC:= SETREC+[DARECNO+1];
                        LEQ : IF DATA^ <= VAL THEN
                                SETREC:=SETREC+[DARECNO+1];
                        GTR : IF DATA^ > VAL THEN
                                SETREC:=SETREC+[DARECNO+1];
                        GEQ : IF DATA^ >= VAL THEN
                                SETREC:=SETREC+[DARECNO+1];
                        INSYM : IF DATA^ IN SETNO THEN
                                SETREC:=SETREC+[DARECNO+1];
                        NOTIN : IF NOT (DATA^ IN SETNO) THEN
                                SETREC := SETREC + [DARECNO+1]
                      END;
                      GET(DATA); DARECNO := DARECNO+1
                    END;
                    CLOSE(DATA,LOCK)
                  END;
          QUOT : BEGIN
                    RESET(STRDATA,DNAME);
                    DARECNO := 0; SETREC := [ ];
                    WHILE NOT EOF(STRDATA) DO
                    BEGIN
                      CASE FCT OF
                      EQL : IF STRDATA^ = QUOTS THEN
                                SETREC:=SETREC+[DARECNO+1];
                      NEQ : IF STRDATA^ <> QUOTS THEN
                                SETREC:=SETREC+[DARECNO+1]
                      END;
                      GET(STRDATA); DARECNO := DARECNO+1
                    END;
                    CLOSE(STRDATA,LOCK)
              END
        END;
      IF (STAT = RETR) OR (STAT = CKEY)
        THEN GETTUPLENO(INSYM,A,B)
      ELSE SETTPL := SETREC
  END ELSE
    IF (STAT = RETR) OR (STAT = CKEY) THEN
      CASE FCT OF
        INSYM : GETTUPLENO(INSYM,A,B);
        NOTIN : GETTUPLENO(NOTIN,A,B)
      END ELSE
        CASE FCT OF
          INSYM : SETTPL := SETREC;
          NOTIN : BEGIN
                    SETTPL := [];
                    FOR J := 1 TO MAXTPL DO
                      IF NOT (J IN SETREC) THEN
                        SETTPL := SETTPL + [J]
```

```
                END
            END;
        K := K-1; TEMPS[K] := SETTPL
    END;

(*************************************************************)
(*                                                           *)
(*  THIS PROCEDURE GIVES THE RECORD NUMBER(S) OF THE         *)
(*  REQUIRED ATTRIBUTE(S) WHICH MEET THE GIVEN CONDITION     *)
(*  THE RESULTS WILL PASS TO THE NEXT SEQUENCE ( SEE         *)
(*  SEQUENCE AND REFERENCE TABLES ) OR BE PRINTED OUT        *)
(*                                                           *)
(*************************************************************)

PROCEDURE QUERY(RNO,ANO: INTEGER);
  VAR J : INTEGER;

  PROCEDURE RELATE1: (* FIND IN RELATION NO. 1 *)
    BEGIN
      RESET(TABLE,RNAME);
      IF (SETTPL = []) AND (I = TOP) THEN
        BEGIN WRITELN('Not found'); EXIT(INTRPRET) END
      ELSE
        FOR J := 1 TO MAXTPL DO
          IF (J IN SETTPL) AND (NOT EOF(TABLE)) THEN
          BEGIN
            SEEK(TABLE,J-1);
            GET(TABLE);
            WITH TABLE^ DO
              BEGIN
                IF (1 IN ATTS) AND (NOT EOF(TABLE)) THEN
                  IF I = TOP THEN
                    BEGIN
                      SEEK(NAMEDATA,NAME-1);
                      GET(NAMEDATA);
                      L := LENGTH(NAMEDATA^);
                      STR := CONCAT(NAMEDATA^,
                                    COPY(BLNK,1,20-L));
                      WRITE(STR)
                    END ELSE SETREC:= SETREC+ [NAME] ;
                IF (2 IN ATTS) AND (NOT EOF(TABLE)) THEN
                  IF I = TOP THEN
                    BEGIN
                      SEEK(DEPTDATA,DEPT-1);
                      GET(DEPTDATA); WRITE(DEPTDATA^ : 4)
                    END ELSE SETREC:= SETREC+ [DEPT];
                IF (3 IN ATTS) AND (NOT EOF(TABLE)) THEN
                  IF I = TOP THEN
                    BEGIN
                      SEEK(SKILLDATA,SKILL-1);
                      GET(SKILLDATA);
                      L := LENGTH(SKILLDATA^);
```

```
                    STR := CONCAT('    ',SKILLDATA^,
                                  COPY(BLNK,1,15-L));
                    WRITE(STR)
                  END ELSE SETREC:= SETREC+ [SKILL];
                IF (4 IN ATTS) AND (NOT EOF(TABLE)) THEN
                  IF I = TOP THEN
                    BEGIN
                      SEEK(SALDATA,SAL-1);
                      GET(SALDATA); WRITE(SALDATA^ : 6)
                    END ELSE SETREC:= SETREC+ [SAL];
                IF (5 IN ATTS) AND (NOT EOF(TABLE)) THEN
                  IF I = TOP THEN
                    BEGIN
                      SEEK(MANNODATA,MANNO-1);
                      GET(MANNODATA);WRITE('    ',MANNODATA^)
                    END ELSE SETREC:= SETREC+ [MANNO]
              END;
              IF I = TOP THEN WRITELN
          END;
        IF I <> TOP THEN
          BEGIN K := K+1; TEMPS[K] := SETREC END;
      CLOSE(TABLE,LOCK)
    END;

  PROCEDURE RELATE2;   (* FIND IN RELATION NO. 2 *)
    BEGIN
      RESET(CHDTBL,RNAME);
      IF (SETTPL = []) AND (I = TOP) THEN
        BEGIN WRITELN('Not found'); EXIT(INTERPRET) END
      ELSE
        FOR J := 1 TO MAXTPL DO
          IF (J IN SETTPL) AND (NOT EOF(CHDTBL)) THEN
          BEGIN
            SEEK(CHDTPL,J-1); GET(CHDTPL);
            WITH CHDTBL^ DO
              BEGIN
                IF (6 IN ATTS) AND (NOT EOF(CHDTBL)) THEN
                  IF  I = TOP THEN
                    BEGIN
                      SEEK(NAMEDATA,NAME-1);
                      GET(NAMEDATA);
                      L := LENGTH(NAMEDATA^);
                      STR := CONCAT(NAMEDATA^,
                                    COPY(BLNK,1,20-L));
                      WRITE(STR)
                    END ELSE SETREC:= SETREC+ [NAME];
                IF (7 IN ATTS) AND (NOT EOF(CHDTBL)) THEN
                  IF I = TOP THEN
                    BEGIN
                      SEEK(CHDDATA,CHNAME-1);
                      GET(CHDDATA);
                      L := LENGTH(CHDDATA^);
```

```
                    STR := CONCAT(CHDDATA^,
                                  COPY(BLNK,1,20-L));
                    WRITE(STR)
                  END ELSE SETREC:= SETREC+ [CHNAME];
                IF (8 IN ATTS) AND (NOT EOF(CHDTBL)) THEN
                  IF I = TOP THEN
                    BEGIN
                      SEEK(SEXDATA,SEX-1);
                      GET(SEXDATA);
                      L := LENGTH(SEXDATA^);
                      STR := CONCAT(' ',SEXDATA^,' ');
                      WRITE(STR)
                    END ELSE SETREC:= SETREC+ [SEX];
                IF (9 IN ATTS) AND (NOT EOF(CHDTBL)) THEN
                  IF I = TOP THEN
                    BEGIN
                      SEEK(AGEDATA,AGE-1);
                      GET(AGEDATA);
                      WRITE(AGEDATA^:3)
                    END ELSE SETREC:= SETREC+ [AGE]
              END;
            IF I = TOP THEN WRITELN
          END;
        IF I <> TOP THEN
          BEGIN K := K+1; TEMPS[K] := SETREC END;
        CLOSE(CHDTBL,LOCK)
      END;

      BEGIN  (* QUERY *)
        IF I = TOP THEN WRITELN;
        IF 1 IN ATTS THEN
          BEGIN RESET(NAMEDATA,'#5:NAME');
            IF I = TOP THEN WRITE('        NAME          ') END;
        IF 2 IN ATTS THEN
          BEGIN RESET(DEPTDATA,'#5:DEPT');
            IF I = TOP THEN WRITE('DEPT') END;
        IF 3 IN ATTS THEN
          BEGIN RESET(SKILLDATA,'#5:SKILL');
            IF I = TOP THEN WRITE('     SKILL        ') END;
        IF 4 IN ATTS THEN
          BEGIN RESET(SALDATA,'#5:SAL');
            IF I = TOP THEN WRITE('  SAL ') END;
        IF 5 IN ATTS THEN
          BEGIN RESET(MANNODATA,'#5:MANNO');
            IF I = TOP THEN WRITE('   MANNO ') END;
        IF 6 IN ATTS THEN
          BEGIN RESET(NAMEDATA,'#5:NAME');
            IF I = TOP THEN WRITE('        NAME          ') END;
        IF 7 IN ATTS THEN
          BEGIN RESET(CHDDATA,'#5:CHNAME');
            IF I = TOP THEN WRITE(' CHNAME            ') END;
        IF 8 IN ATTS THEN
```

```
            BEGIN RESET(SEXDATA,'#5:SFX');
              IF I = TOP THEN WRITE(' SFX ') END;
          IF 9 IN ATTS THEN
            BEGIN RESET(AGEDATA,'#5:AGE');
              IF I = TOP THEN WRITE('AGE') END;
          WRITELN; WRITELN;
          RNAME := CONCAT('#5:',RELTBL[RNO].NAME5);
          BLNK := '                    '; SETREC := [];
            CASE RNO OF
              1 : RELATE1;
              2 : RELATE2
            END;
          IF 1 IN ATTS THEN CLOSE(NAMEDATA,LOCK);
          IF 2 IN ATTS THEN CLOSE(DEPTDATA,LOCK);
          IF 3 IN ATTS THEN CLOSE(SKILLDATA,LOCK);
          IF 4 IN ATTS THEN CLOSE(SALDATA,LOCK);
          IF 5 IN ATTS THEN CLOSE(MANNODATA,LOCK);
          IF 6 IN ATTS THEN CLOSE(NAMEDATA,LOCK);
          IF 7 IN ATTS THEN CLOSE(CHDDATA,LOCK);
          IF 8 IN ATTS THEN CLOSE(SEXDATA,LOCK);
          IF 9 IN ATTS THEN CLOSE(AGEDATA,LOCK)
      END;

    PROCEDURE DEL(A,B,C : INTEGER; D : STRING);
                            (* DELETE THE ASSOCIATED INDEX FROM *)
                            (*     THE RELATIONSHIPS FILE       *)
      VAR P : INTEGER;
      BEGIN
        RESET(ANYREL,D);
        INVTBL[C+A] := INVTBL[C+A] - [B];
        ANYREL.SETNO := INVTBL[C+A];
        SEEK(ANYREL,A-1); PUT(ANYREL);
        CLOSE(ANYREL,LOCK)
      END;  (* DELETE *)

    PROCEDURE PUTS(A,B,C : INTEGER; D : STRING);
                            (* PUT THE ASSOCIATED INDEX IN THE *)
                            (*    RELATIONSHIPS FILE           *)
      VAR P : INTEGER;
      BEGIN
        RESET(ANYREL,D);
        INVTBL[C+A] := INVTBL[C+A] + [B];
        ANYREL.SETNO := INVTBL[C+A];
        SEEK(ANYREL,A-1); PUT(ANYREL);
        CLOSE(ANYREL,LOCK)
      END;  (* PUTS *)




    PROCEDURE SETREL(RNO,ANO,I : INTEGER); FORWARD;
```

```
(*************************************************************)
(*                                                           *)
(*  THIS PROCEDURE WILL UPDATE THE DATA BASE BY UPDATING     *)
(*  THE INDEX NUMBER(S) IN THE RELATIONSHIPS FILE(S).        *)
(*                                                           *)
(*************************************************************)

PROCEDURE UPDATE(VAL,RNO,ANO,T : INTEGER; RNAME,FNAME,
                 QUOTS : STRING; OBJ : SYMBOL; ST : STATUS);
  VAR  I, J, K : INTEGER;
  BEGIN
    IF NOT NEXT THEN
      BEGIN
        DAMECNO := 0;
        CASE OBJ OF
          NBR : BEGIN
                  RESET(ANYDATA,FNAME);
                  IF ANYDATA^ = VAL THEN I := DARECNO+1
                    ELSE REPEAT GET(ANYDATA);
                           DARECNO := DARECNO+1
                         UNTIL (ANYDATA^ = VAL) OR
                               (EOF(ANYDATA));
                  IF NOT EOF(ANYDATA) THEN I := DARECNO+1
                    ELSE BEGIN
                      WRITELN('No such entry found. Insert?');
                      EXIT(INTERPRET)
                    END;
                  CLOSE(ANYDATA,LOCK)
                END;
          QUOT : BEGIN
                  RESET(STRDATA,FNAME);
                  IF STRDATA^ = QUOTS THEN I := DARECNO+1
                    ELSE REPEAT GET(STRDATA);
                           DARECNO := DARECNO+1
                         UNTIL (STRDATA^ = QUOTS) OR
                               (EOF(STRDATA));
                  IF NOT EOF(STRDATA) THEN I :=DARECNO+1
                    ELSE BEGIN
                      WRITELN;'No such entry found. Insert?');
                      EXIT(INTERPRET)
                    END;
                  CLOSE(STRDATA,LOCK)
                END
        END;
      END ELSE
        FOR J := 1 TO MAXTPL DO
          IF J IN SETREC THEN I := J;
      SETTPL(RNO,ANO,I);
      IF T = TOP THEN WRITELN('Updating is done')
    END;  (* UPDATE *)
```

```
PROCEDURE SETTPL;
  BEGIN
    CASE RNO OF
      1 : BEGIN
            RESET(TABLE,RNAME);
            WITH ATTPL[ANO] DO
              BEGIN
                RELNAME := CONCAT(RNAME,'.',ATN);
                K := ADDR
              END;
            FOR J := 1 TO MAXTPL DO
              IF (J IN SETTPL) AND (NOT EOF(TABLE)) THEN
                BEGIN
                  SEEK(TABLE,J-1); GET(TABLE);
                  WITH TABLE^ DO
                    CASE ANO OF
                      1 : NAME := I;
                      2 : BEGIN
                            DEL(DEPT,J,K,RELNAME);
                            DEPT := I; PUTS(DEPT,J,K,RELNAME)
                          END;
                      3 : BEGIN
                            DEL(SKILL,J,K,RELNAME);
                            SKILL := I; PUTS(SKILL,J,K,RELNAME)
                          END;
                      4 : BEGIN
                            DEL(SAL,J,K,RELNAME);
                            SAL := I; PUTS(SAL,J,K,RELNAME)
                          END;
                      5 : BEGIN
                            DEL(MANNO,J,K,RELNAME);
                            MANNO := I; PUTS(MANNO,J,K,RELNAME)
                          END
                    END;
                  SEEK(TABLE,J-1); PUT(TABLE)
                END;
            CLOSE(TABLE,LOCK)
          END;
      2 : BEGIN
            RESET(CHDTBL,FNAME);
            WITH ATTPL[ANO] DO
              BEGIN
                RELNAME := CONCAT(FNAME,'.',ATN);
                K := ADDR
              END;
            FOR J := 1 TO MAXTPL DO
              IF (J IN SETTPL) AND (NOT EOF(CHDTBL)) THEN
                BEGIN
                  SEEK(CHDTBL,J-1); GET(CHDTBL);
                  WITH CHDTBL^ DO
                    CASE ANO OF
                      6 : BEGIN
```

```
                                    DEL(NAME,J,K,RELNAME);
                                    NAME := I; PUTS(NAME,J,K,RELNAME)
                                  END;
                             7 : BEGIN
                                    DEL(CHNAME,J,K,RELNAME);
                                    CHNAME := I; PUTS(CHNAME,J,K,RELNAME)
                                  END;
                             8 : BEGIN
                                    DEL(SEX,J,K,RELNAME);
                                    SEX := I; PUTS(SEX,J,K,RELNAME)
                                  END;
                             9 : BEGIN
                                    DEL(AGE,J,K,RELNAME);
                                    AGE := I; PUTS(AGE,J,K,RELNAME)
                                  END
                           END;
                        SEEK(CHDTEL,J-1); PUT(CHDTEL)
                      END;
                  CLOSE(CHDTEL,LOCK)
                END
            END
        END;  (* STREL *)


        BEGIN  (* INTERPRET *)
          NEXT := FALSE;
          I := 0; J := 0; K := 0;
          REPEAT
            I := I+1; SYO := SYOTBL[I];
            CASE SYO OF
              COND   : BEGIN  (* IDENTIFY THE QUALIFICATION *)
                         J := J+1; INS := REFTBL[J];
                         WITH INS DO
                         BEGIN
                           ST := ATTBL[ANO].STAT;
                           OBJ := ATTBL[ANO].KIND;
                           IF OBJ = VAR THEN
                             CONDITION(RNO,ANO,VAL,ST,PCT,OBJ,' ')
                           ELSE
                             CONDITION(RNO,ANO,*,ST,PCT,OBJ,QUOTS)
                         END
                       END;
              SELSYM : BEGIN                              (* INQUIRY *)
                         SYTBL := TEMPS[K]; J := J+1;
                         WITH REFTBL[J] DO
                           BEGIN ATTS := NAMES; QUERY(RNO,ANO) END
                       END;
              SETSYM : BEGIN                              (*  UPDATE  *)
                         SYTBL := TEMPS[K]; J := J+1;
                         INS := REFTBL[J];
                         WITH INS DO
                           BEGIN
```

```
                                RNAME := CONCAT('#5:',RELTBL[RNO].NAMED);
                                FNAME := CONCAT('#5:',ATTBL[ANO].ATN);
                                ST := ATTBL[ANO].STAT;
                                OBJ := ATTBL[ANO].KIND;
                                IF OBJ = NBR THEN
                                  UPDATE(VAL,RNO,ANO,I,RNAME,FNAME,' ',
                                         NBR,ST)
                                ELSE UPDATE(0,RNO,ANO,I,RNAME,FNAME,
                                            QUOTS,QUOT,ST)
                              END
                          END;
                ORSYM   : BEGIN K := K-1;              (*  OR  *)
                            TEMPS[K] := TEMPS[K] + TEMPS[K+1]
                          END;
                ANDSYM  : BEGIN K := K-1;              (*  AND  *)
                            TEMPS[K] := TEMPS[K] * TEMPS[K+1]
                          END;
                LOD     :  BEGIN (* PASS THE RESULTS TO THE NEXT STEP *)
                            SETREC := TEMPS[K]; K := K-1;
                            NEXT := TRUE
                          END
              END
        UNTIL I = TOP
      END;  (* INTRPRET *)
    END.    (* INTERPRETER *)
```

```
(*****************************************************************)
(*                                                               *)
(*    THIS IS THE MAIN PROGRAM OF THE MICRODATABASE SYSTEM.      *)
(*  THERE ARE FOUR UNITS ( CREATE, PERSER, INTERPRETER, HELP )   *)
(*  TO PERFORM THE THREE FUNCTIONS OF THIS SYSTEM. THE THREE     *)
(*  FUNCTIONS ARE : 1. DATA BASE INITIALIZATION                  *)
(*                  2. DATA BASE MANIPULATION                    *)
(*                  3. HELP FUNCTION                             *)
(*                                                               *)
(*****************************************************************)

PROGRAM MICRODATA;

  USES PARSE, INTERPRETER, CREATE, HELPS;

  VAR  CH : CHAR;

  BEGIN
    RESET('X.'CONSOLE:'); WRITELN;
    WRITELN('MICRO DATA BASE SYSTEM'); WRITELN;
    WRITE('Command : C(reate, X(ecute, H(elp ->');
    READ(X,CH);
    FOR I := 1 TO 2 DO WRITELN;
      CASE CH OF
        'C' : CREATES; (* DATA BASE INITIALIZATION *)
        'H' : HELP   ; (* HELP FUNCTION          *)
        'X' : BEGIN    (* DATA BASE MANIPULATION  *)
                INITIALIZE; PREPARE;
                CX := 0; CC := 0; LL := 0;
                WHILE NOT EOF(X) DO
                  BEGIN
                    SCANNER;
                    STATEMENT;
                    IF (NOERR) AND (SYM = PRD) THEN INTRPRET
                  END
              END
      END
  END.   (* MICRODATA *)
```

BIBLIOGRAPHY

1. Shillirgton, K. and Ackland G.M., UCSD Pascal Version 1.5, Institute for Information System, UCSD mailcode C-021, La Jolla Ca 92093, Sept. 1978.

2. Martin, J., Computer Data-Base Organization, Prentice Hall, Inc., Englewood Cliffs, New Jersey 07632, 1977.

3. Morn, Granino A., Microprocessors and Small Digital Computer Systems For Engineers and Scientists, McGraw-Hill Book Co., 1977.

4. Kroenke, D., Database Processing, Science Research Associates, Inc., 1977.

5. Chamberlin, D.D., Relational Data-Base Management Systems, Computing Surveys, Vol. 8, No. 1, March 1976, pp 43-66.

6. Digital Research, Post Office Box 579, Pacific Grove Ca 93950, CP/M Newsletter, August, 1979.

7. Date, C.J., An Introduction To Database Systems, 2d ed., Addison Wesley Publishing Co., 1977.

8. Astrahan, M.M. and others, System R: Relational Approach to Database Management, ACM Trans. on Database Systems, Vol. 1, No. 2, June 1976, pp 97-137.

9. Chamberlin, D.D. and others, SEQUEL2: A Unified Approach to Data Definition, Manipulation and Control, IBM Journal Research and Development, November 1976, pp 560-575.

10. Cardenas, A.F., Data Base Management Systems, Allyn and Bacon, Inc., Boston, Massasuchetts, 1977.

INITIAL DISTRIBUTION LIST

		No. copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 52Bz Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
4.	LCDR P.Purkhead, USN, Code 52Pg Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
5.	LT M. Moranville, USN, Code 52My Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
6.	MAJ Amrun Sehan, The Indonesian Navy Mess Doser PUSDIKIAL SESKOABRI BAG. LAUT Cipulir, Kebayoran Lama, Jakarta Selatan, Indonesia	1
7.	MAJ T.M. Sihombing, The Indonesian Army DISPULLAHTA TNI-AD Jl. Veteran No. 5 Jakarta, Indonesia	1
8.	DISPULLAHTA TNI-AD Jl. Veteran No. 5 Jakarta, Indonesia	1
9.	DISPULLAHTA TNI-AL Wisma Lumba-lumba, Jl. Gatotsubroto No.110 Jakarta Pusat, Indonesia	1
10.	PUSDIKIAL SESKOABRI BAG. LAUT Cipulir, Kebayoran Lama, Jakarta Selatan, Indonesia	1

11. LTC T. Rumbayan, SMC# 2106 — 1
Department of Computer Science
Naval Postgraduate School
Monterey, California 93940

12. LTC Sudiyarto, The Indonesian Navy — 1
Kotak Pos 4/KBYCL
Jakarta Selatan, Indonesia

13. LTC Yoon C. Sup, The Republic Of Korea Army — 1
#15-205, KUZ-DONG-APT.
SU-CHO-DONG, KANG-NAM-KU,
Seoul, Korea